



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Hochschule für Technik und Wirtschaft

Fachbereich Wirtschaftswissenschaften II

Studiengang Angewandte Informatik

Masterarbeit

**Konzeption und Prototyp einer
intelligenten Arbeitssuche nach
persönlichen Fähigkeiten**

vorgelegt von Sergej Meister
Matrikelnummer: s0521159

Erstgutachter: Prof. Dr. Christian Herta

Zweitgutachter: Prof. Dr.-Ing. habil. Dierk Langbein

Berlin, 15. Februar 2016

Vorwort

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Ziele der Arbeit	2
1.3 Aufbau der Arbeit	3
2 Grundlagen	4
2.1 Dokumentenorientierte Datenbanken	4
2.2 IntelliJob	7
2.2.1 Technologie Stack	9
2.2.2 Software Architektur	10
2.2.3 Problembeschreibung	11
2.3 Tag Cloud	11
2.4 Informationsbeschaffung	12
2.4.1 Qualität in Information Retrieval System	13
2.4.2 Indexierung von Textdokumenten	14
2.4.3 Boolesche Suche	15
2.4.4 Vektorraummodell	15
2.4.5 Volltextsuche	16
2.4.6 Gewichtete Suche	17
2.4.7 Autovervollständigung	18
2.5 Elasticsearch	18
2.5.1 Sharding	20
3 Systementwurf	22
3.1 OpenNLP oder Elasticsearch	22
3.2 Prozessablauf	22
3.3 Datenstruktur	23

3.4 Schichtenarchitektur	23
4 Entwicklung und Implementierung	24
4.1 Elasticsearch - Konfiguration	24
4.2 Benutzerprofile	24
4.2.1 Modell	25
4.2.2 Controller	25
4.2.3 Services	25
4.2.4 View	26
4.3 Volltextsuche	26
4.3.1 Modell	26
4.3.2 Controller	26
4.3.3 Services	27
4.3.4 View	27
4.4 Gewichtete Suche	27
4.4.1 Modell	28
4.4.2 Controller	28
4.4.3 Services	28
4.4.4 View	28
4.5 Autovervollständigung	29
4.5.1 Modell	29
4.5.2 Controller	29
4.5.3 Services	30
4.5.4 View	30
5 Evaluierung	31
5.1 Volltext-Suchanfragen	32
5.2 Gewichtete Suchanfragen	43
5.3 Vergleich und Fazit	46
6 Problemen und Schwierigkeiten	48
7 Zusammenfassung und Ausblick	49
Literaturverzeichniss	i
Abbildungsverzeichnis	ii

Tabellenverzeichnis	iii
Glossar	iv
Anhang	v
Eigenständigkeitserklärung	xviii

1. Einleitung

1.1 Motivation

Was versteht man unter dem Begriff *Information*? Was sind *Daten*? Die beiden Fragen sind sehr beliebt im Informatikstudium, besonders im ersten Semester. Doch die Informationen und Daten spielen nicht nur in der Informatik sondern auch in andren Gebieten eine wichtige Rolle. Täglich treffen die Menschen eine Entscheidung auf der Basis ihres Wissensdaten, aus denen Informationen extrahiert werden. Dadurch dass Informationen aus den Daten gelesen werden, hängt ein Entscheidungsfaktor wesentlich von der „*Qualität*“ den bereitgestellten Daten ab.

In der heutigen Informationsgesellschaft durch das schnelle Wachstum von digital verfügbaren Daten ist es für die Menschen schwieriger geworden, die entscheidungsrelevante Informationen aus den Datenmengen zu filtern. Deswegen bietet das Internet viele verschiedene Suchportals für unterschiedliche Interessengruppe, die eine gezielte Suche nach bestimmten Datenmengen erlauben.

Die Jobportals unterstützen ihre Anwender rund um das Thema „*Arbeitsuche*“ und „*Bewerbungsprozess*“. Ein Arbeitssuchender legt Suchkriterien, wie Berufsbezeichnung und Region, fest und bekommt innerhalb weniger Sekunden eine Liste von Arbeitsangeboten, die den Suchkriterien entsprechen. Es ist offensichtlich, dass die Arbeitssuche dank Jobportals viel effizienter und einfacher geworden ist. Trotzdem muss der Arbeitssuchender immer noch viele Informationen selbst raus finden und interpretieren. Zum Beispiel für die Entscheidung, ob die Stelle den gewünschten Anforderungen entspricht, sind die Daten wichtig, wie Berufsname, Firmenname und Qualifikation. Für das Bewerbungsschreiben werden noch weitere Daten benötigt, wie Kontaktperson, Mailadresse, Firmenhomepage und Firmenadresse. Diese Informationen muss der Arbeitssuchender ständig selbst raus finden. Genau mit dieser Problematik beschäftigt sich das Programm „*IntelliJob*“, das diese Daten automatisch aus dem Arbeitsangebot extrahiert.

Das Programm „*IntelliJob*“ wurde im Rahmen der Veranstaltung „*Informationssystemen*“ unter Betreuung von Professor Christian Herta entwickelt und kann bereits Berufsbezeichnung, Mailadresse, Firmenhomepage, Kontaktperson und Firmenadresse automatisch auslesen und in Datenbank speichern. Außerdem verfügt das Programm über eine Weboberfläche, um die gespeicherte Daten tabellarisch darzustellen.

Der weitere Nachteil von den Jobportals ist die Implementierung der Suche, die in der Profile gespeicherten personalisierten Daten nicht berücksichtigt. Denn die typische Suchkriterien eines Jobportals sind einfache Stichworte wie Softwareentwickler, Java, C# u.s.w. Daraus wird es ersichtlich, dass die Suche nach einem konkreten Beruf erfolgt. Dieses Suchverfahren eignet sich gut für Quereinsteiger und für die Menschen, die auf der Suche nach neuer Berufsorientierung sind, aber nicht für die Menschen, die eine Arbeit entsprechend ihren Fähigkeiten und Qualifikationen suchen. Wäre es nicht besser, wenn die Suche nach persönlichen Fähigkeiten und Qualifikationen erfolgte, die von Benutzer selbst bewertet werden? Nehmen wir an, der Mensch bewertet seine Java-Kenntnisse mit 2 und C# mit 5 Sternen. Dann ist es doch logisch, dass alle C# Arbeitsangebote in der Ergebnisliste zuerst angezeigt werden müssen. Vielleicht ist es auf dem ersten Blick nicht ganz ersichtlich, woran die beide Verfahren sich unterscheiden. Die Suche nach einem Beruf als Softwareentwickler kann aber die Java- Arbeitsangebote zuerst anzeigen, was aus der Sicht eines Arbeitssuchender mit den besseren C# Kenntnissen nicht ganz zutreffend ist. Der weitere Unterschied liegt im Entscheidungsfaktor. Bei dem ersten Verfahren entscheidet der Mensch, welche Arbeit er haben will. Bei dem zweiten Verfahren entscheidet das Programm, welche Arbeit für den Mensch am besten geeignet ist.

Die Untersuchung von mehreren Jobportals wie „Xing“, „StepStone“, „JobScout24“, „Monster“ hat bestätigt, dass es tatsächlich kein deutsches Jobportal gibt, das die Suche nach persönlichen Fähigkeiten und Qualifikationen anbietet.

1.2 Ziele der Arbeit

Im Rahmen dieser Masterarbeit wird ein Suchverfahren implementiert, dass die Suche nach persönlichen Fähigkeiten und Qualifikationen unterstützt. Die Suche muss mit *Elasticsearch*¹ realisiert und in die Anwendung *IntelliJob* integriert werden. Um die Suchqualität bewerten zu können, müssen mehrere unterschiedliche Suchanfragen gebildet und genau untersucht werden. Am Ende der Arbeit wird geprüft, ob das neue Suchverfahren gegenüber der traditionellen Suche nach Stichworte ein besseres Ergebnis liefert. Das Suchverfahren wird als effektiv bewertet, wenn die erste Arbeitsangebote in der Ergebnisliste den Benutzeranforderungen besser entsprechen werden.

Das weitere Thema dieser Arbeit wird die Datenbeschaffung sein. Um nach persönlichen Fähigkeiten und Qualifikationen suchen zu können, müssen diese Daten zuerst erfasst werden. Dafür wird das Projekt „IntelliJob“ um eine neue Web-Form erweitert, die die Bewertung von persönlichen Fähigkeiten und Qualifikationen ermöglicht. Die Daten, die dem Arbeitssuchender bereitgestellt werden, müssen sinnvoll gruppiert werden, so dass die Kriterien wie Java und C# unter den Sammelbegriffen Programmieren oder Softwareentwickeln gefunden werden könnten. Der Arbeitssuchender muss in der Lage sein, seine eigene Daten speichern, editieren

¹www.elastic.co.

und löschen zu können. Außerdem darf der Benutzer selbst entscheiden, welches Suchverfahren eingesetzt werden soll.

1.3 Aufbau der Arbeit

Der schriftliche Teil der Arbeit beschreibt ein kompletter Entwicklungsprozess. Im Kapitel „*Grundlagen*“ werden die Technologie erläutert, die für die Entwicklung neues Suchverfahrens relevant sind. Dabei wird das Projekt *IntelliJob* und die eingesetzte Technologie kurz vorgestellt. Das Kapitel „*Systementwurf*“ befasst sich mit der Softwarearchitektur und beschreibt die Logikverteilung und die Datenkommunikation innerhalb der Anwendung. Das vierte Kapitel „*Entwicklung und Implementierung*“ stellt das Prototyp des neuen Suchverfahrens vor. Dabei wird sowohl das Datenmodell als auch die wichtigste Methoden detailliert beschrieben. Das Kapitel besteht aus mehrere Unterkapiteln. Jedes Unterkapitel beschreibt die Implementierung von einem konkreten Anwendungsfall. Das Kapitel „*Evaluierung*“ analysiert mehrere unterschiedliche Suchanfragen und bewertet sie. Problemen und Schwierigkeiten, die während der Implementierung aufgetreten sind, sowie die Art und Weise, wie diese Probleme gelöst oder eventuell nicht gelöst wurden, werden in dem 6. Kapitel erfasst. Zum Schluss wird die Arbeit zusammengefasst und ein Resümee gezogen. Ein Ausblick zeigt auf, welche Erweiterungen für die Anwendung in Zukunft noch sinnvoll sein könnten.

2. Grundlagen

In diesem Kapitel wird auf die allgemeinen und informatischen Grundlagen eingegangen, welche zum Verständnis dieser Masterarbeit relevant sind. Die Anwendung *IntelliJob* persistiert bereits die Daten in eine dokumentenorientierte Datenbank *MongoDB*¹ und es wird noch eine weitere dokumentenorientierte Technologie *Elasticsearch* eingesetzt. Deswegen ist es wichtig mindestens die Grundlagen sowie die Vor- und Nachteile von dokumentenorientierten Datenbanken im Blick zu behalten. Die in der Arbeit verwendete *Elasticsearch-Feature's* werden in einem extra Kapitel ausführlich beschrieben. Da die Suchfunktionalität in die bereits vorhandene Anwendung integriert werden muss, ist es für die Bewertung der Arbeit notwendig den IST-Zustand zu dokumentieren.

2.1 Dokumentenorientierte Datenbanken

Bei dokumentenorientierten Datenbanken werden die Daten in Form von Dokumenten gespeichert. Die typische Dokument-Formate sind XML, YAML und JSON. Die einzelnen Datenfelder werden als Key-Value Stores zusammengefasst. Jedes Dokument ist vollkommen frei bezüglich seine Struktur und das ist ein wesentlicher Unterschied zu relationalen Datenbanken, wo die Datenstruktur vordefiniert ist [Dorschel 2015]. Die Abbildung 2.1 zeigt eine typische normalisierte Datenstruktur in der relationalen Welt. Einige Datensätze in der Tabelle *skills* beinhalten kein "*description*". Die Spalte muss in dem Fall einfach leer bleiben. Es besteht keine Möglichkeit nur für bestimmte Datensätze auf die Spalte "*description*" zu verzichten.

category_id	name	skill_id	category_id	name	description
1	Languages	1	1	German	
2	Knowledges	2	1	English	
3	Personal strengths	3	2	Java	programming language java
		4	2	PHP	programming language php

Abbildung 2.1: Relationale Tabellen *skill-categories* und *skills*

In dokumentenorientierten Datenbanken ist es aber durchaus möglich, dass einige Dokumente innerhalb einer *Collection* kein *description* Feld haben. Das sieht man deutlich in der *Collection skills* auf der nächsten Seite.

¹www.mongodb.org.

Mongo Collection *skill-categories*

```
{
  "_id": "569ab65644ae6344028b6cb7",
  "name": "Languages"
},
{
  "_id": "569ab65644ae6344028b6cb7",
  "name": "Knowledges"
},
{
  "_id": "569ab66d44ae6344028b6cb9",
  "name": "Personal strengths"
}
```

Mongo Collection *skills*

```
{
  "_id": "569ab85144ae6344028b6cbf",
  "category": {
    "_id": "569ab61a44ae6344028b6cb5",
    "name": "Languages"
  },
  "skills": [
    {
      "_id": "569ab8dd44ae6344028b6cc0",
      "name": "deutsch"
    },
    {
      "_id": "569ab8dd44ae6344028b6cc1",
      "name": "english"
    }
  ]
},
{
  "_id": "569ab9d644ae6344028b6cc9",
  "category": {
    "_id": "569ab65644ae6344028b6cb7",
    "name": "Knowledges"
  },
  "skills": [
    {
      "_id": "569ab9d644ae6344028b6cca",
      "name": "java",
      "description": "programming language java"
    }
  ]
}
```

Wenn man die beide *Collection* genau anschaut, fehlt es sofort auf, dass die *Category-Daten* nicht redundant sind. In den relationalen Datenbanken werden die Datenattribute durch die Anwendung von verschiedenen Normalisierungsregeln so lange zerlegt, bis keine vermeidbaren Redundanzen mehr vorhanden sind. Die dokumentenorientierte Datenbanken verfolgen den Prinzip der *Aggregation*. Der Begriff kommt eigentlich aus dem Domain-Driven Design und bedeutet, dass jedes Dokument eine eigenständige Einheit ist, das von keinen anderen beziehungsweise von möglichst wenigen Dokumenten abhängig ist. Dadurch können die Daten immer vollständig automar gespeichert oder gelesen werden.

Diese hohe Datenflexibilität ist aber nicht umsonst. Zum Ersten ist es oft aufwendig die Daten zu ändern, die bereits in anderen Dokumenten gespeichert sind. Ein Beispiel wäre dafür das Dokument mit dem Name „*Languages*“ in der *Collection* „*skill-categories*“ mit einem neuen Attribut zu erweitern oder komplett zu löschen. In dem Fall muss diese „*Category*“ auch in allen anderen Dokumenten gefunden und entsprechend geändert werden. Zum Zweiten können dokumentenorientierte Datenbanken aufgrund ihrer Schemafreiheit keine einfache Datenvalidierung der Attributen und Datentypen durchführen. Das muss stattdessen im Programmcode behandelt werden, was zu höheren Softwarekosten führt und fehleranfällig ist [Jan 2013].

Wenn man das ganze kurz wörtlich zusammenfasst: Schemafreiheit, keine Datenvalidierung, mehr Programmcode, Fehleranfällig und aufwändige Datenänderung auf der Datenbankebene, fragt man sich, wieso werden aber die dokumentenorientierte Datenbanken, die seit 2008 unter dem Titel „*NoSQL*“² bekannt sind, immer beliebter? Eine einfache Antwort lautet, dass die *NoSQL*-Datenbanken oft schneller als relationale Datenbanken sind. Statt mehrere Tabellen mit *JOIN*-Operation abfragen zu müssen, führt ein *NoSQL*-Datenbank nur eine einzige Abfrage aus, um ein kompletter Datensatz zu erhalten. Die Datenkonsistenz beim Lesen und Schreiben in mehrere Tabellen wird in klassischen relationalen Datenbanken durch Transaktion-Mechanismen realisiert. Jede Transaktion ist isoliert von einander und wird ganz oder gar nicht ausgeführt. Wenn eine Transaktion in eine Tabelle schreibt, kann keine andere Transaktion auf die Tabelle zugreifen, bis der Schreibvorgang komplett abgeschlossen oder unterbrochen wird. Das führt dazu, dass die Lese- und Schreiboperation von parallel ausgeführten Transaktionen einander blockieren und warten müssen, bis alle nötige Ressource freigegeben werden. Dadurch dass *NoSQL* Datenbanken keine Abfrage blockieren, werden die auch schneller abgearbeitet. Es besteht zwar die Möglichkeit, dass inkonsistente Daten gelesen werden, aber aufgrund, dass die Daten oft nur aus einer einzigen *Collection* gelesen werden, ist die Fehlerwahrscheinlichkeit enorm klein im Vergleich zu viel steigender Performance[Dorschel 2015]. Die einzige Schutzmaßnahme, die von meisten *NoSQL*-Datenbanken unterstützt wird, ist die Datenversionierung. Bei jedem Speicherzugriff wird die Version des Dokumentes geprüft, ist die Version identisch dem bereits gespeicherten Dokument, dann wird ein *Exception* geworfen. Denn es ist ein Zeichen dafür, dass Dokument bereits von einem anderen *Request* geändert

²<https://de.wikipedia.org/wiki/NoSQL>.

wurde. Außerdem wird auch die Datenversionierung für die Synchronisierung zwischen mehrere Rechnerknoten innerhalb eines *Clusters* verwendet. Das *Sharding Concepts* in dokumentenorientierten Datenbanken wird im Kapitel über *Elasticsearch* detailliert beschrieben.

Fazit: Die dokumentenorientierte Datenbanken eignen sich besonders gut für die Anwendungen, wo hohe Geschwindigkeit bei Datenzugriffe und Datenverarbeitung im Vordergrund steht und die Datenredundanz nicht hoch priorisiert wird. Die meisten Datenbankzugriffe in der Anwendung *IntelliJob* sind Lesezugriffe. Es gibt wenige Möglichkeiten die Daten durch Benutzeraktion zu ändern. Die Änderungen beeinflussen oft nur ein einziges *Collection* und meistens sogar nur ein einziges Dokument. Deswegen kann eine dokumentenorientierte Datenbank die Anwendungsanforderungen viel besser erfüllen.

2.2 IntelliJob

Die Anwendung „*IntelliJob*“ wurde bereits mehrmals erwähnt und muss endlich vorgestellt werden. Es gab mal eine Idee die bewerbungsrelevante Daten aus Arbeitsangeboten automatisiert zu extrahieren. Diese Idee wurde mit dem Programm „*IntelliJob*“ realisiert. Da die Anwendung im Rahmen der Veranstaltung „*Informationssystemen*“ eigenständig entwickelt und dokumentiert wurde, werden einige Inhalte aus der Arbeit in diesem Kapitel zitiert.

Bevor die Daten aus dem Arbeitsangebot gelesen werden könnten, muss dieses Arbeitsangebot erstmals ermittelt werden. Dafür werden mehrere Job-Agenten auf Jobportals *Stepstone*³ und *Monster*⁴ angelegt, die die Arbeitsangebote per Email täglich senden. Das Email beinhaltet mehrere HTML-Links mit unterschiedlichen Arbeitsangeboten. Die Anwendung „*IntelliJob*“ holt diese Emails aus dem Postfach ab, ruft die Links auf und speichert das geladene Inhalt in Datenbank ab.

Im nächsten Schritt werden die bewerbungsrelevante Daten, *Berufsbezeichnung*, *Firmenadresse*, *Firmenhomepage*, *Mailadresse* und *Kontaktperson*, aus dem Arbeitsangebot extrahiert.

Berufsbezeichnung wird einfach aus dem HTML-Link ausgelesen, das als *Value* immer eine Berufsbezeichnung enthält.

Mailadresse und **Firmenhomepage** werden mit Hilfe von Regulären Ausdrücken extrahiert.

Mail-Pattern: `[a-zA-Z0-9_+.]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-]+`

WWW-Pattern: `www.[\w\d.:#@%/$()~_?+&]*`

HTTP-Pattern: `http.[\w\d.:#@%/$()~_?+&]*`

³www.stepstone.de.

⁴www.monster.de.

HTTPS-Pattern: `https.[\w\d.:#@%/_;$()~_?+-=&]*`

Leider können Reguläre Ausdrücke nur Daten mit einer eindeutigen Struktur ermitteln. **Kontaktperson** und **Firmenadresse** können aber sehr unterschiedlich geschrieben werden:

- Alle der Kosmonauten 26a D-32451 Bad Kreuznach
- Straße des 17. Juni 17a 13456 Berlin (Lieblings Beispiel)
- Herr Richard-Alexander Wagner
- Johann Wolfgang von Goethe
- Alexander Sergejewitsch Puschkin

Aus den oben aufgelisteten Beispielen wird es ersichtlich, dass es kaum möglich ist, allein mit Regulären Ausdrücke, diese Daten zu extrahieren. Deswegen werden diese Daten in „IntelliJob“ mit Hilfe von „Apache OpenNLP Framework“⁵ ermittelt und zusammen mit allen anderen bewerbungsrelevanten Daten in Datenbank gespeichert.

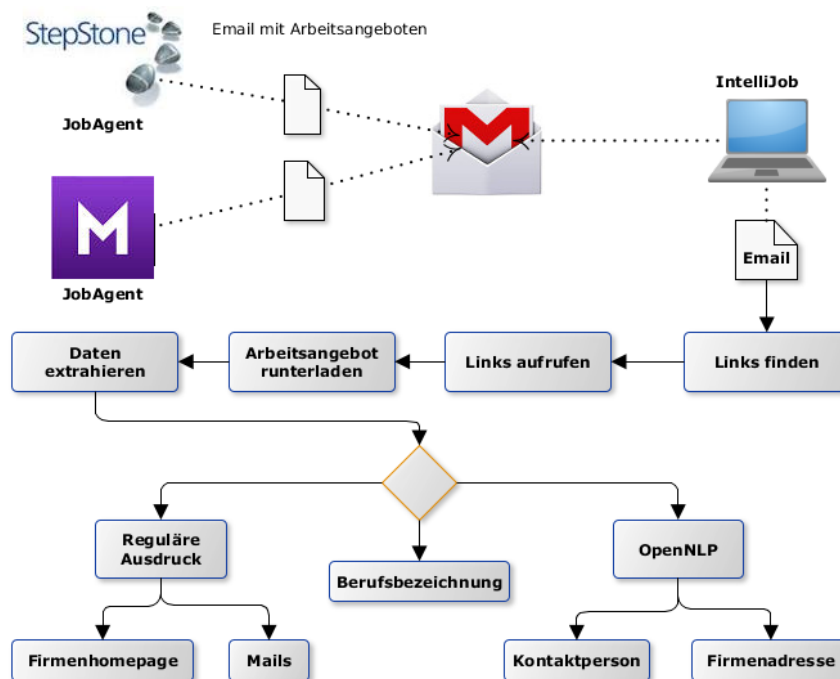


Abbildung 2.2: Prozessablauf in IntelliJob

⁵<https://opennlp.apache.org>.

2.2.1 Technologie Stack

Apache OpenNLP ist ein Open Source Produkt und wird als Maven-Dependency in der Version 1.5.3 in das Projekt „*IntelliJob*“ eingebunden. Das Framework setzt die Technologie aus dem Gebiet *Natural Language Processing* ein, um bestimmte Bestandteile eines Textes zu erkennen und zu klassifizieren. Die Muster-Erkennung basiert auf künstlich erzeugten Trainingsdaten, die durch ein Lernprozess in einem Model gespeichert werden. Das Framework wird ständig weiter entwickelt und verfügt über eine gute Dokumentation.⁶

Die Anwendungslogik von „*IntelliJob*“ wird in der Programmiersprache Java geschrieben und verwendet sehr viele Komponenten aus dem Spring Framework⁷. Spring Framework ist ebenfalls ein Open Source Produkt, welches performante und moderne Java-Enterprise Einsätze realisiert. Mit dem Einsatz von Spring *Dependency Injection* und *Aspekt orientierte Programmierung* wird das Programmcode redundant verteilt und kann auch leicht wiederverwendet werden. Alle erzeugte Objekte werden anhand der *Spring Data*⁸ in eine dokumentenorientierte Datenbank MongoDB persistiert.

Die weitere Besonderheit von „*IntelliJob*“ ist, dass es eine Standalone Web-Anwendung ist, die lokal auf dem Client einen embedded Web Container *Tomcat*⁹ startet. Der große Vorteil von diesem Einsatz liegt darin, dass der Webserver ein Teil der Anwendung ist, und die Anwendung nicht mehr in den Webserver deployed werden muss. Das Programm wird als JAR-Datei ausgeliefert und kann direkt in der Java Virtual Machine ausgeführt werden. Realisiert wird das ganze mit *Spring Boot*¹⁰ Technologie, die dank Autokonfiguration sowohl alle nötige Resource zusammenpackt, als auch alle Rest-Services zur Kommunikation mit Frontend initialisiert.

Das Frontend wird mit *AngularJS*¹¹ umgesetzt und verfügt bereits über 6 verschiedene Views. Das Framework unterstützt die Entwickler bei der Implementierung von Single Page Anwendungen. Während der Masterarbeit wird *AngularJS* von der Version 1.2.16 zu 1.4.7 aktualisiert.

Home	Startseite
Emails	zeigt alle gefundene Emails von Job Portals
Job Links	zeigt alle in Mails gefundene Links zu Arbeitsangeboten
Jobs	zeigt alle runter geladene Arbeitsangebote
Jobs Details	zeigt alle extrahierte Daten eines Arbeitsangebotes
Audit	dient zur Datenauswertung und zeigt sowohl die aktuellste als auch die alte Ergebnisse des Datenextraktions.

⁶<https://opennlp.apache.org/documentation/1.6.0/manual/opennlp.html>.

⁷www.spring.io.

⁸<http://projects.spring.io/spring-data>.

⁹<http://tomcat.apache.org>.

¹⁰<http://projects.spring.io/spring-boot>.

¹¹<https://angularjs.org>.

2.2.2 Software Architektur

„IntelliJob“ ist quelloffen und kann auf Versionsverwaltungsplattform, GitHub¹², heruntergeladen werden. Die Anwendungslogik basiert auf Schichtenarchitektur, die sequenziell von oben nach unten abgearbeitet wird.

Präsentationsschicht repräsentiert die Daten in HTML-Form und regelt die Interaktionen zwischen Benutzer und Software. Alle Benutzeranfragen werden asynchron an Serviceschicht weitergeleitet. Die Daten zwischen beiden Schichten werden in JSON-Format übertragen.

Serviceschicht nimmt die Client-Anfragen entgegen, validiert Benutzerdaten und wandelt sie in Domain-Objekte um. Danach wird die Abfrage der Controlschicht übergeben. Letztendlich wird ein *Response* generiert und zurückgesendet.

Controlschicht vereint alle Methoden aus Datenzugriffsschicht und führt alle nötige Operationen aus, um die Benutzeranfrage zu bearbeiten.

Datenzugriffsschicht kapselt die Zugriffe auf persistente Daten. Die Datenaustausch erfolgt über Domain-Objekte.

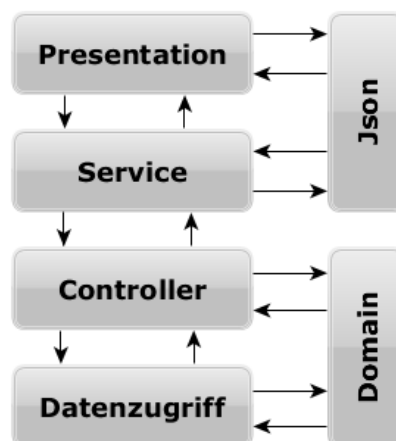


Abbildung 2.3: Schichtenarchitektur

Dadurch dass die Anwendungslogik in mehrere Schichten verteilt wird, wird die Komplexität der Anwendung wesentlich reduziert, was sowohl für das Verständnis als auch für die Wartung von Software ein großer Vorteil ist. Außerdem können die Programmteile dank ihrer Abstraktion gut getestet und auch leicht wiederverwendet werden.

Um die Methoden zur Datenextraktion auch in anderen Projekten zu verwenden, werden sie in ein separates Projekt, namens *Civis-Tools*¹³ ausgelagert.

¹²<https://github.com/SergejMeister/intellijob>.

¹³<https://github.com/SergejMeister/civis-tools>.

aufgerufen wurde, wird der Schlagwort der Tag Cloud höher gewichtet[OnPage 2016].

Die Tag Cloud wird meistens nach der folgenden Formel berechnet:

$$S_i = [(f_{max} - f_{min}) * \frac{t_i - t_{min}}{t_{max} - t_{min}} + f_{min}]$$

- S_i - anzuzeigende Schriftgröße
- f_{max} - maximale Schriftgröße
- f_{min} - minimale Schriftgröße
- t_i - Häufigkeit des betreffenden Schlagwortes
- t_{min} - Häufigkeit, ab der ein Schlagwort angezeigt werden soll
- t_{max} - Häufigkeit des häufigsten Schlagwortes¹⁴

2.4 Informationsbeschaffung

Im ersten Kapitel wurde schon erwähnt, dass die Informationen aus Daten extrahiert werden. Zur Informationsbeschaffung (eng. Information Retrieval) gehören neben Extraktionsprozess noch weitere Prozesse. Zuerst müssen die Daten erfasst werden und in einer Datenbank oder einem Daten-Management-System in einer bestimmten Syntax abgelegt werden. Dabei legt der Verfasser einen Teil seines Wissen in ein Dokument nieder, das in Text, Bild oder eine andere Form gespeichert wird. Im nächsten Schritt müssen die Daten, die bereits existieren, wiedergefunden und in eine geeignete Repräsentationsform gebracht werden. Letztendlich sollen nur die Informationen extrahiert werden, die möglichst gut bei einer Problemlösung helfen können. Der Autor Andreas Heinrich in seinem Lehrtext *„Information Retrieval Grundlagen, Modelle und Anwendungen“* bezeichnet dieses Prozess als Wissenstransfers und weist deutlich darauf hin: *„... Information Retrieval nicht nur als den Prozess der Wissensextraktion aus einer Dokumentenkollektion sondern als Prozess des Wissenstransfers zu verstehen“*[Henrich 2008]. Die Abbildung 2.7 stellt das Prozess vereinfacht dar.

Das System, das sich mit der Informationsbeschaffung befasst, wird in der Informatik als IR-System bezeichnet. Die Aufgabe des IR-Systems besteht hauptsächlich darin, die hinter der Anfrage stehende Informationsbedürfnisse zu befriedigen. Oft muss ein IR-System neben der Anfrage noch weitere Datenquellen, wie Feedback oder vorhandene Information über Nachfragender, zu berücksichtigen, um eine bessere Informationsergebnis zu erzielen. Im Bezug auf die Anwendung *„IntelliJob“* kann Informationsergebnis wesentlich verbessert werden, indem der Nachfragender die Suchanfrage selber priorisiert.

¹⁴<https://de.wikipedia.org/wiki/Schlagwortwolke>.



Abbildung 2.5: Prozess des Wissenstransfers vom Autor eines Dokumentes bis zum Informations-Nachfragenden

2.4.1 Qualität in Information Retrieval System

Um ein gutes IR-System von einem schlechten unterscheiden zu können, muss die Qualität der Ergebnissen gemessen werden. Die Qualität wird ihrerseits durch **Relevanz**, **Recall** und **Precision** bestimmt.

Relevanz kann binär (*relevant/nicht relevant*) oder mit Relevanzeinstufung (von 1 bis 10) definiert werden. In der Praxis hat sich allerdings die binäre Relevanz durchgesetzt. Die Entscheidung, ob ein Dokument relevant oder nicht relevant ist, ist sehr subjektiv und erfordert Vorwissen sowohl über das untersuchte Dokument als auch über andere Dokumenten. Deswegen kann es sehr schwierig sein alle Dokumente einzustufen[Henrich 2008].

Recall beschreibt der Anteil der relevanten Dokumenten in der Ergebnismenge im Verhältnis zu allen relevanten Dokumenten und wird wie folgt berechnet:

$$Recall = \frac{a}{S}$$

- a - Anzahl der relevanten Dokumente im Ergebnis

- S - Gesamtzahl der relevanten Dokumente

Precision zeigt die Genauigkeit der Ergebnismenge.

$$Precision = \frac{a}{D}$$

- a - Anzahl der relevanten Dokumente im Ergebnis
- D - Gesamtzahl der Dokumente im Ergebnis

Die oben beschriebene Vorgehensweise zur Bestimmung von *Recall* und *Precision* Werten eignet sich gut für unsortierte Ergebnismenge, wo es hauptsächlich darum geht möglichst viele relevante Dokumente zu finden. In der heutigen Informationsgesellschaft kann eine Ergebnisliste extrem viele Dokumente beinhalten und alle diese Dokumente können auch unterschiedlich relevant sein. Am Interessantesten sind deswegen nur die erste k -Dokumente der Ergebnisliste. Für die Evaluierung von sortierter Liste, auch Ranking Liste genannt, wird eine Folge von *Recall* und *Precision* Werten betrachtet, die für jedes Dokument einzeln berechnet werden. Die Vorgehensweise ist folgendes:

- Zuerst muss eine Testmenge mit k -Dokumenten zur Evaluierung festgelegt werden.
- Wenn ein Dokument aus der Testmenge in der Ergebnisliste gefunden wird, werden die *Recall* und *Precision* Werte berechnet. Grundsätzlich muss es auch gelten: $Recall(k+1) > Recall(k)$ und $Precision(k+1) > Precision(k)$
- Wenn ein Dokument nicht gefunden wird, bleibt die *Recall* Wert unverändert. Die *Precision* Wert muss aber kleiner werden: $Recall(k+1) = Recall(k)$ und $Precision(k+1) < Precision(k)$

Laut der Definition kann jetzt für einen *Recall* Wert mehrere *Precision* Werte geben. Um es linear auszugleichen, wird der höchsten *Precision* Wert für jeden *Recall* Wert genommen. Das wird als *interpolated precision* bezeichnet[Manning u. a. 2008, S.158-159]

2.4.2 Indexierung von Textdokumenten

Die vorliegende Arbeit beschäftigt sich mit Arbeitsangeboten, die in Textform vorliegen. Diese Textdokumente müssen automatisiert erfasst werden. Dieses Prozess wird in Information Retrieval als *Indexierung* bezeichnet. Bei der Indexierung wird ein Dokument analysiert und alle Wörter außer *Stopwörter* werden ermittelt. Die *Stopwörter* sind besondere Wörter die sehr häufig vorkommen. In der deutschen Sprache sind Artikeln ein typisches Beispiel für die *Stopwörter*. Die extrahierte Worte werden in Grundform (z.B. *schreiben* - *schreib*) gebracht und als Terme abgespeichert. Durch Grundformredaktion wird es verhindert, dass die gleiche aber unterschiedlich geschriebene Worte mehrmals gezählt werden. Denn während der

Indizierung wird sowohl das Vorkommen eines Wortes in einem Dokument (term frequency, TF) als auch in allen anderen Dokumenten (inverse document frequency, IDF) gezählt.

$$IDF_i = \log \frac{N}{n_i}$$

- i - der untersuchte Term
- N - Anzahl aller Dokumenten
- n_i - Anzahl der Dokumente mit Term

Der Logarithmus sorgt dafür, dass ein oft vorkommender Term nicht extrem hoch gewichtet wird. Das Endergebnis der automatischen Indexierung ist ein Gewicht für jeden Term in jedem Dokument [Mandl 2001]. Nachdem das Dokument erfolgreich indexiert wurde, kann es auch durchgesucht werden.

2.4.3 Boolesche Suche

Die Boolesche Suche oder das Boolesche Retrieval Modell basiert auf die elementare Mengenoperationen, wie Schnittmenge (AND-Verknüpfung), Vereinigungsmenge (OR-Verknüpfung) und Komplementärmenge (NOT-Verknüpfung). Das Modell geht davon aus, dass das Ergebnis einer Suchanfrage als exakte Menge von relevanten Dokumenten bestimmt werden kann. Dabei werden nur Dokumente ausgegeben, die den in der Suchanfrage formulierten booleschen Ausdruck erfüllen.

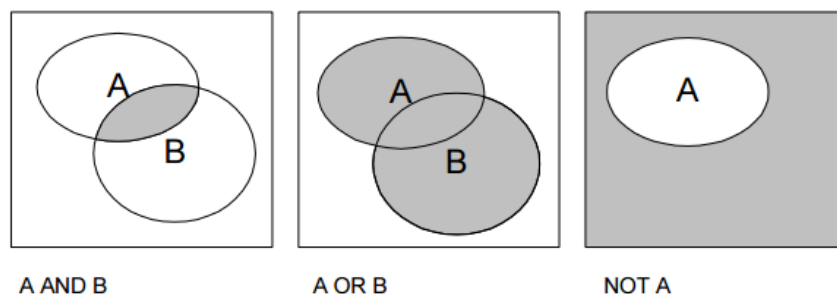


Abbildung 2.6: Die elementaren booleschen Operationen

Der Nachteil des booleschen Modells besteht darin, dass die Gewichtung von Termen nicht berücksichtigt wird und die Ergebnismenge deswegen ungeordnet ausgeliefert wird. Außerdem werden viele Dokumente nicht gefunden, weil die gesuchte Elemente nicht in den exakt gleichen Form in Dokumenten enthalten sind. Trotz seiner Ungenauigkeit wird das boolesche Model oft eingesetzt vor allem um mehrere Suchanfragen mit einander zu verknüpfen.

2.4.4 Vektorraummodell

Zum ersten Mal wurde das Vektorraummodell von Gerard Salton, Andrew Wong und Chungshu Yang in einem Zeitschriftartikel unter dem Thema „A Vector Space

Model for Automatic Indexing" im Jahr 1975 publiziert und ausführlich beschrieben [Salton u. a. 1975]. Die Grundidee dieses Modells besteht darin, dass sowohl die Anfrage als auch alle Dokumente in einem vieldimensionalen Vektorraum dargestellt werden. Dabei bildet jeder einzelne Term eine Achse beziehungsweise eine Dimension in diesem Vektorraum und die Koordinaten werden durch die Gewichtung von Termen repräsentiert.

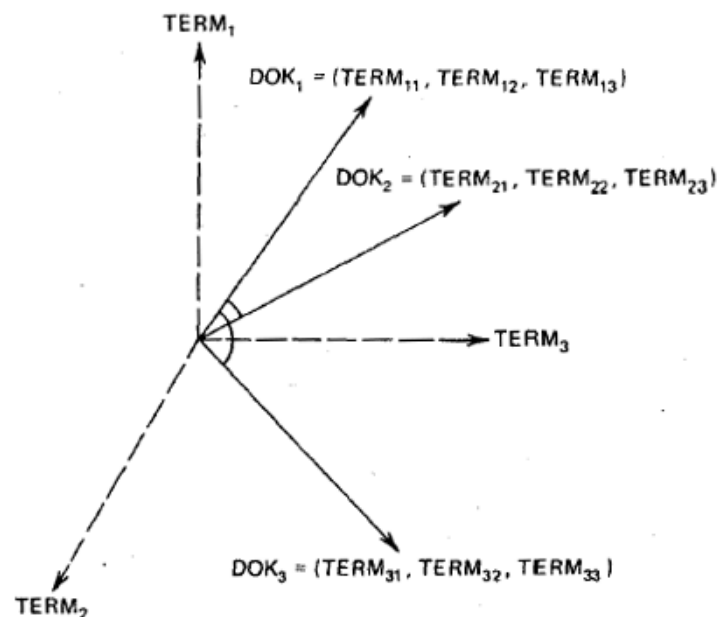


Abbildung 2.7: Vektorraumpräsentation eines Dokumentraums (Salton u. McGill 1987, 129)

Die Relevanz eines Dokumentes im Vektorraummodell wird oft durch Berechnung des Cosinus des Winkels zwischen Anfrage und Dokument bestimmt. Das führt dazu, dass nicht mehr nach exakten Übereinstimmungen sondern nach der Ähnlichkeit zwischen Suchanfrage und Dokumenten gesucht wird und die Ergebnismenge in dem Fall auch sortiert ausgegeben werden kann.

Obwohl das Vektorraummodell die wesentlichen Nachteile des booleschen Modells aufhebt, kann das boolesche Modell unter bestimmten Bedingungen eine bessere Ergebnismenge als Vektorraummodell liefern. Damit die Ähnlichkeitsmaß zwischen Anfrage und Dokument mit dem Vektorraummodell sinnvoll berechnet werden könnte, muss die Anfrage aus mehreren Begriffen bestehen. Wenn die Suchanfrage aus wenigen Begriffen besteht, kann das boolesche Modell, das diese Begriffe einfach mit *AND* verknüpft, oft zu besseren Ergebnissen führen.

2.4.5 Volltextsuche

Bei der Volltextsuche handelt es sich darum, eine benutzerdefinierte und strukturierte Suchanfrage in einem unstrukturierten Textdokument zu finden. Natürlich kann man dafür das boolesche oder Vektorraummodell einsetzen, um die erste

Ergebnisse schnell zu bekommen. Allerdings bieten die meisten IR-System noch einige Optimierungsmethode, die die Ergebnismenge wesentlich beeinflussen können. Der nachfolgende Abschnitt beschreibt kurz die Optimierungsmethode, die für den praktischen Teil nützlich sein könnten.

Phrasensuche wird für die Suche nach zusammenhängenden Begriffen eingesetzt. Abhängig von Einsatzszenarien kann die Phrasensuche entweder sehr gute oder sehr schlechte Ergebnismenge liefern. Wenn man nach einen Title oder eine exakte Phrase in einem Dokument sucht, dann bekommt man gute Ergebnisse. Allerdings werden viele potenziell relevante Dokumente ausgeschlossen, weil die gesuchte Phrase in diesen Dokumenten nicht in der gleichen Wortfolge vorliegt. Sucht man beispielsweise nach „*relationale Datenbanken*“, werden Dokumente mit „*relationale Datenbank*“ nicht gefunden.

Fuzzy-Suche sorgt für Fehlertoleranz und erlaubt eine vordefinierte Abweichung zwischen den gesuchten und in Dokument vorhandenen Term. Das bekannteste Algorithmus für die Fuzzy-Suche ist Levenshtein-Distanz¹⁵. Das Algorithmus berechnet die Anzahl von Änderungen, die nötig sind, um ein Wort in ein anderes umzuwandeln. Je wenig Änderungen notwendig sind, desto ähnlicher sind die Worte.

Filter werden verwendet, um die Suchergebnisse nach bestimmten Kriterien einzuschränken. Der meistverbreitete Filter ist *Seiten-Pagination*. Die andere Einsatzmöglichkeiten sind Zeit- oder Typ-Filter.

2.4.6 Gewichtete Suche

In dem Kapitel „*Indexierung von Textdokumenten*“ wurde bereits erwähnt, dass jeder Term von IR-System während der Indexierung gewichtet wird. Allerdings werden nicht nur Terme sondern auch die Suchanfragen(eng. search query) von IR-System gewichtet. Wenn die Suchanfrage aus mehreren einzelnen Suchanfragen besteht, dann wird die Gewichtung für alle einzelnen Anfragen ebenfalls berechnet.

Während die Volltextsuche die bessere Ergebnismenge durch Manipulation von Termen erreichen will, mischt sich die gewichtete Suche direkt in der Berechnung der Gewichtung ein. Dieses Verfahren ist als *Boosting* bekannt und wird meistens zur Anzeige von beliebtesten oder populärsten Dokumenten verwendet.

Es gibt drei unterschiedliche *Boosting*-Verfahren: **Field-Boosting**, **Term-Boosting** und **Query-Boosting**. **Field-Boosting** wird eingesetzt, wenn die Suche mehrere Felder anfragt und ein bestimmtes Feld höher als anderen priorisiert werden soll. Beim **Term-Boosting** wird die Suchanfrage durch Vorkommen eines bestimmten Terms höher gewichtet. Mit der **Query-Boosting**, wie der Name schon sagt, kann eine *Query* geboosted werden.

¹⁵<https://de.wikipedia.org/wiki/Levenshtein-Distanz>.

2.4.7 Autovervollständigung

Autovervollständigung ist heutzutage fast in jeder Anwendung vorhanden. Die Möglichkeit, durch die Eingabe von wenigen Zeichen auf vermutlich relevanter Suchbegriff zu kommen, hat sich in der Realität als sehr praktisch erwiesen. Grundsätzlich basiert dieses Verfahren auf die Ergänzung von eingegebenen Anfangsbuchstaben. Trotzdem gibt es immer mehr Anwendungen, wo die eingegebene Buchstaben nicht als Anfangsbuchstaben sondern als einen Suchbegriff interpretiert werden um vermutlich relevante Dokumente sofort zur Auswahl anzuzeigen.

Mit Hilfe von Autovervollständigung wird das Erinnerungsproblem zu einem Wiedererkennungsproblem. Denn es reicht wenige Informationen, um ein gewünschter Suchbegriff zu treffen. Außerdem können auch Tippfehler bei der Autovervollständigung berücksichtigt und vermieden werden.

2.5 ElasticSearch

ElasticSearch ist ein Open-Source Suchserver, welcher auf Apache Lucene Bibliothek basiert. Das Projekt wird in der Programmiersprache Java geschrieben und ermöglicht die Kommunikation über REST-Schnittstelle. Die Daten werden in JSON-Format übertragen und in einem invertierten Index abgespeichert. Ein Index entspricht einer Datenbank. Jeder Datensatz wird durch Dokumenten und Spalten durch Felder repräsentiert. ElasticSearch verfolgt eine dokumentenorientierte Architektur und ist deswegen in der Lage auch komplexe Objekte in einem Feld zu speichern.

Relationale Datenbank	Elasticsearch
Datenbank	Index
Tabelle	Dokumenttype
Datensatz	Dokument
Spalte	Feld

Tabelle 2.1: Begriffsvergleich zwischen Relationale Datenbanken und ElasticSearch

ElasticSearch unterstützt alle im Kapitel 2.4 vorgestellte Methoden der Informationsbeschaffung und erweitert sie mit neuer Funktionalität. Zum Beispiel ist es möglich die Dokumente in Echtzeit zu indexieren. Die Indexierung wird dabei im Arbeitsspeicher gehalten und erst später auf die Festplatte geschrieben. Besonders mächtig ist ElasticSearch bei der Berechnung von Gewichtung(eng. score). Denn es werden mehrere Funktionen zur Manipulation von Score bereitgestellt. Es ist sogar erlaubt eigene Funktion zur Berechnung von Score zu schreiben. Standardmäßig wird aber Score nach Lucene-Formel berechnet¹⁶.

¹⁶www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html.

$$score(q, d) = queryNorm(q) * coord(q, d) * \sum_{t \in (in)q} (tf * idf^2 * boost(t) * norm(t, d))$$

- $score(q, d)$ - Relevanz eines Dokumentes (d) für die Suchanfrage (q)
- tf - term frequency (Kapitel 2.4.2) - $tf = \sqrt{termFreq}$
- idf - inverse document frequency (Kapitel 2.4.2) - $idf = 1 + \log(\frac{numDocs}{docFreq+1})$
- $queryNorm(q)$ - Funktion zur Berechnung der Normalisierung von *Search-Query* - $queryNorm(q) = 1/\sqrt{sumOfSquaredWeights}$, $sumOfSquaredWeights$ summiert IDF von allen Termen in der Suchanfrage
- $coord(q, d)$ - wird verwendet, um Dokumente mit mehrere vorhandenen Suchtermen höher zu gewichten $coord(q, d) = \frac{t(in)q}{t(in)d}$
- $boost(t)$ - erlaubt *boosting* von *Search-Query* (Kapitel 2.4.6). Standardmäßig ist die Wert gleich 1.
- $norm(t, d)$ - Funktion zur Berechnung von Feldnormalisierung
 $norm(t, d) = 1/\sqrt{numTerms}$, $numTerms$ steht für die Anzahl von Termen im gesuchten Feld

Eine wichtige Funktion, die in diesem Zusammenhang auch erwähnt werden soll, ist die *Decay*-Funktion. *Decay*-Funktion berechnet die Relevanz eines Dokumentes abhängig von der benutzerdefinierten Distanz zwischen einem numerischen Feldwert und dem Original-Wert. Das numerische Feld kann eine Zahl, ein Datum oder Geo-Location sein. Die Distanz wird *linear*, *exponential* oder standardmäßig nach *Gaus*-Algorithmus berechnet.

$$S(doc) = \exp(-\frac{\max(0, |fieldValue - origin| - offset)^2}{2\sigma^2})$$

- $fieldValue$ - Feldwert
- $origin$ - Startwert, der bei der Distanzberechnung berücksichtigt wird. Beim Datum kann es zum Beispiel das heutiges Datum sein.
- $offset$ - Decay-Funktion wird berechnet, wenn die Distanz größer als $offset$. Standardwert ist 0.
- σ^2 -sorgt dafür, dass die berechnete Distanz in einem benutzerdefinierten Intervall liegt. $\sigma^2 = -\frac{scale^2}{2*\log(decay)}$, $decay$ definiert ein Abstand zwischen den Dokumenten. Standardwert ist 0.5. $scale$ ist die benutzerdefinierte Distanz.

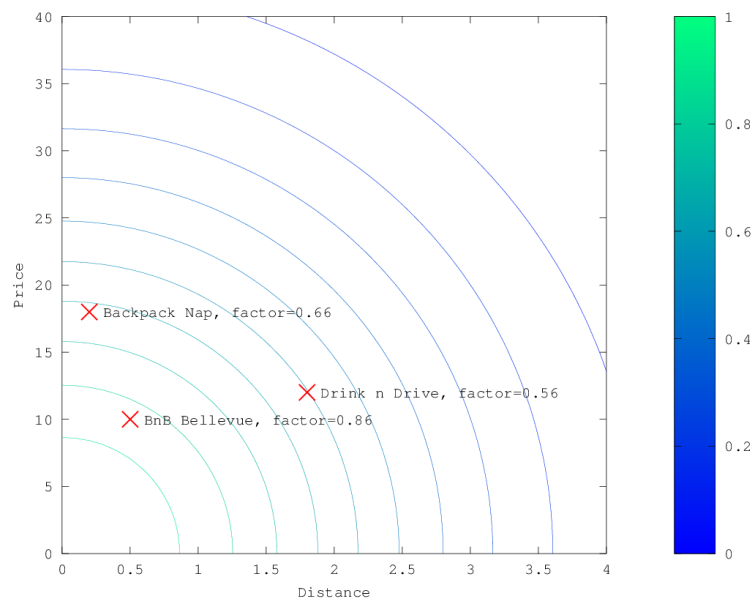


Abbildung 2.8: Visualisierung von *Decay*-Funktion mit *Gaus*

2.5.1 Sharding

Das Wort *Elastic* in dem Projektname bezieht sich auf horizontale Skalierbarkeit, welche ermöglicht, die Daten auf mehrere Server zu verteilen. Alle Daten werden in einem Cluster vorgehalten. Ein Cluster besteht aus einem oder mehreren Knoten (eng. Node), die auf unterschiedlichen Server laufen können. Die Synchronisierung zwischen den Knoten wird über Cluster gesteuert und kann relativ einfach und flexibel von Benutzer konfiguriert werden. Wenn ein Server besonders gut ausgerüstet ist, ist es sinnvoll ein *Master*- und ein *Slave*-Node festzulegen.

Um Ausfallsicherheit zu gewährleisten, werden die Dokumente der einzelnen Indizes in mehrere *Shards* repliziert und auf den Knoten verteilt. Der *Shard*, welcher das Dokument zuerst speichert, wird *primary shard* genannt. Alle weitere *shards* sind *replica shards* und behalten nur die Kopie von diesem Dokument. Wenn ein *Node* ausfällt, kann das Dokument aus dem *replica shard* eines anderen Nodes gelesen werden[Rafal Kuc 2013]. Die Abbildung 2.9 stellt ein Cluster mit zwei Knoten, vier *primary shard* und ein *replica shard* dar.

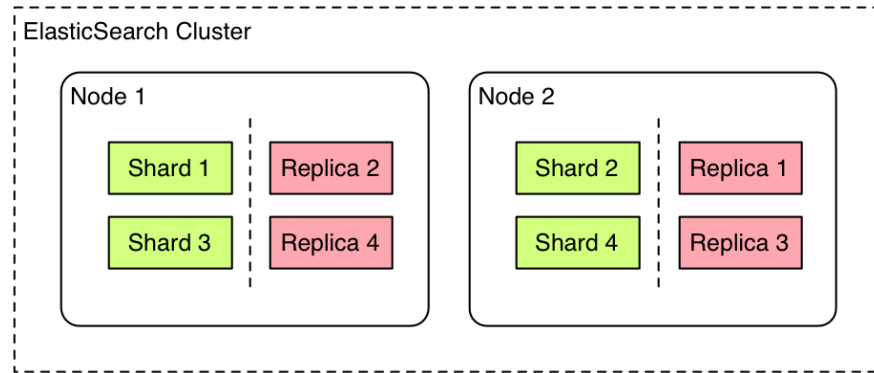


Abbildung 2.9: Beispiel - Ein Cluster mit zwei Knoten, 4 *shards* und 1 *replica shard*[Terrier 2013]

3. Systementwurf

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.1 OpenNLP oder ElasticSearch

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.2 Prozessablauf

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdspra-

chige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.3 Datenstruktur

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3.4 Schichtenarchitektur

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4. Entwicklung und Implementierung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.1 Elasticsearch - Konfiguration

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.2 Benutzerprofile

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdspra-

chige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.2.1 Modell

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.2.2 Controller

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.2.3 Services

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.2.4 View

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.3 Volltextsuche

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.3.1 Modell

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.3.2 Controller

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest

gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.3.3 Services

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.3.4 View

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.4 Gewichtete Suche

Die gewichtete Suchanfragen werden mit *Boosting* realisiert. Wenn ein Suchbegriff zutrifft, wird die *Query*, die diesen Suchbegriff enthält, mit einem bestimmten Faktor geboosted. Damit ändert sich die Score und auch die Relevanz dieses Dokumentes gegenüber des anderen Dokumenten.

4.4.1 Modell

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.4.2 Controller

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.4.3 Services

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.4.4 View

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest

gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.5 Autovervollständigung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.5.1 Modell

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.5.2 Controller

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdspra-

chige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.5.3 Services

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4.5.4 View

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

5. Evaluierung

Während der Masterarbeit wurden mehrere *Search-Queries* implementiert, die aber aufgrund ihrer schlechten Effektivität in der Anwendung nicht angewendet werden. Da alle diese *Queries* durch schrittweise Optimierung von Nachteilen zu der Endergebnis geführt haben, werden auch diese *Queries* in diesem Kapitel evaluiert und detailliert beschrieben.

Um einzelne *Query* mit einander effektiv vergleichen zu können, müssen zwei Voraussetzungen erfüllt werden. Die Evaluierung muss genau auf die *Query* beschränkt werden und die Datenmenge muss unverändert bleiben. Der einzige Grund für Datenänderung kann nur durch inkonsistente Datenmenge entstehen, die zur falschen Evaluierungsergebnissen führt. In der Tabelle unten wird deswegen der Datenausgangszustand kurz zusammengefasst.

Anzahl Dokumenten	Ältestes Datum	Neuestes Datum	Analyzer
3144	14.03.2015	11.10.2015	standard

Tabelle 5.1: Datenausgangszustand für die Evaluierung

Wie in dem vorherigen Kapitel schon erwähnt wurde, wird die Implementierung von allen *SearchQueries* in statische Methode separat ausgelagert und über eine Utility-Klasse(*SearchQueryUtility*) zur Verfügung gestellt. Für die Evaluierung werden diese Methode einzeln ausgeführt und die resultierte *SearchQuery* zusammen mit Pagination-Daten an *ElasticSearch*-Client direkt übergeben. Um die *SearchQuery* besonders die *Score*-Berechnung genau untersuchen zu können, bittet *ElasticSearch* durch *Explain*-Mode eine sehr detaillierte Ergebnisausgabe. Mit dem *SpringData-Framework* und *ElasticSearch-API* kann es wie folgt realisiert werden.

```
PageRequest pageRequest = new PageRequest(0, 50);
SearchQuery searchQuery = SearchQueryUtility.buildFullTextSearchBoolQuery_1(
    "Werkstudent Java, Datenbanken", pageRequest);
SearchResponse searchResponse = elasticSearchConfiguration.getNode()
    .client().prepareSearch("intellijob").setTypes("jobDetails")
        .setQuery(searchQuery.getQuery())
        .setExplain(true)
        .setSize(50)
        .get();
```

5.1 Volltext-Suchanfragen

Query Nr. 1

```
{
  "bool": {
    "should": [
      {
        "match": {
          "content": {
            "query": "Werkstudent Java",
            "type": "boolean",
            "operator": "AND"
          }
        }
      },
      {
        "match": {
          "content": {
            "query": "Datenbanken",
            "type": "boolean"
          }
        }
      }
    ],
    "minimum_should_match": "1"
  }
}
```

Beschreibung: Es gibt zwei boolesche Suchanfragen, die zu einer gemeinsamen ebenfalls boolescher Suchanfrage zusammengefasst sind. Die Suchbegriffe sind mit *AND* verknüpft. „*minimum_should_match : 1*“ sorgt dafür, dass entweder die erste oder die zweite Suchanfrage treffen muss, damit das Dokument relevant wird.

#	Score	Arbeitsbezeichnung	Datum
1	0.13931298	WERKSTUDENT SOFTWAREENTWICKLUNG IM BEREICH BAHNTECHNIK	21.09.2015 11-24
2	0.13591066	Werkstudent im Bereich Software Entwicklung - Smart HomeBosch Software Innovations GmbH	28.04.2015 02-50
3	0.13591066	Werkstudent im Bereich Software Entwicklung - Smart HomeBosch Software Innovations GmbH	28.04.2015 02-50

Tabelle 5.2: Query Nr. 1 - Ergebnis-Ausschnitt

Ergebnis: Zur Evaluierung vom ersten *Query* werden nur drei erste Ergebnisse angezeigt, die schon deutlich darauf hinweisen, dass die Datenmenge nicht konsis-

tent ist. Es gibt also mehrere gleiche Arbeitsangebote, die auch gleich gewichtet werden.

Query Nr. 1.1

Nachdem die duplizierte Arbeitsangebote gelöscht wurden, sind nur **2292** Arbeitsangebote übrig geblieben.

Ergebnis: Die Tabelle unten zeigt die ersten fünf Arbeitsangebote an.

#	Score	Arbeitsbezeichnung	Datum
1	0.19563562	Werkstudent im Bereich Software Entwicklung - Smart HomeBosch Software Innovations GmbH - Berlin	16.08.2015 02-29
2	0.1758025	Werkstudent im Bereich Softwareentwicklung (Java)	23.08.2015 02-39
3	0.16529423	WERKSTUDENT SOFTWAREENTWICKLUNG IM BEREICH BAHNTECHNIK	21.09.2015 11-24
4	0.15650849	Werkstudent im Bereich Frontend Design	19.09.2015 03-13
5	0.15650849	Werkstudent im Bereich Software Development Toolchain	18.04.2015 02-53

Tabelle 5.3: Query Nr. 1.1 - Ergebnis-Ausschnitt

Um jede Änderung von *Query* effektiv nachvollziehen und bewerten zu können, werden immer die gleiche Suchbegriffe „Softwareentwickler Java, Testautomatisierung“, für allen Volltextsuchanfragen verwendet. In der Tabelle(5.4) wird die Relevanz von einzelnen Suchbegriffen zusammengefasst.

Suchbegriff	relevante Dokumente
Softwareentwickler	149
Java	305
Softwareentwickler {AND} Java	73
Testautomatisierung	22
Softwareentwickler {AND} Java {AND} Testautomatisierung	6

Tabelle 5.4: Relevanz von Suchbegriffen für die Volltextsuche

Aus der Tabelle(5.4) wird es ersichtlich, dass die exakte Suche nach „Softwareentwickler Java, Testautomatisierung“ insgesamt 89 (73+22-6) relevante Arbeitsangebote liefern soll.

Query Nr. 1.2

```
{
  "bool": {
    "should": [
      {
        "match": {
          "content": {
            "query": "Softwareentwickler Java",
            "type": "boolean",
            "operator": "AND"
          }
        }
      },
      {
        "match": {
          "content": {
            "query": "Testautomatisierung",
            "type": "boolean"
          }
        }
      }
    ],
    "minimum_should_match": "1"
  }
}
```

Ergebnis:

#	Arbeitsbezeichnung	Datum
1	Softwareentwickler RefactoringInterhyp	15.07.2015
2	Softwareentwickler/ -architekt	09.07.2015
3	Senior Java-Entwickler Angebotsimport	01.10.2015
4	Senior Java-Entwickler Full Stack	14.05.2015
5	Senior Software Developer (Java/Backend)	21.09.2015

Tabelle 5.5: Query Nr. 1.2 - Ergebnis- 5 beste Hits

Die Tabelle(5.5) zeigt die ersten fünf Dokumenten der Ergebnisliste. Insgesamt werden genau 89 Arbeitsangebote gefunden. Wenn diese Zahl in die Formel zur Berechnung von *Recall* und *Precision* aus dem Kapitel 2.4.1 eingesetzt wird, kommt als Ergebnis raus:

Ergebnis(<i>D</i>)	Rel. in Ergebnis (<i>a</i>)	Rel. in Dok(<i>S</i>)	Recall	Precision
89	89	89	1 = 100%	1 = 100%

Tabelle 5.6: Query Nr. 1.1 - falsch-positive Recall, Precision

Auf den ersten Blick scheint die *Query* echt positiv zu sein. Denn die Relevanz eines Dokumentes bei der booleschen Suchanfrage wird allein durch das Existenz von gesuchten Termen bestimmt. Wenn man aber die Ergebnismenge aus der Tabelle(5.5) unter Berücksichtigung des Datums von zuletzt gespeicherten Arbeitsangebot am 11.10.2015 genau anschaut, fehlt es auf, dass einige Arbeitsangebote sehr veraltet und aus der Benutzersicht nicht mehr relevant sind.

In Wirklichkeit bei der großen Datenmenge und bei der großen Anzahl von relevanten Dokumenten interessiert es den Benutzer nicht, wieviele Dokumente gefunden werden, sondern viel mehr wie relevant die ersten 10 oder 20 Dokumente sind. Für die Evaluierung von sortierter Ergebnisliste werden *Recal* und *Precision* Werte, wie im Kapitel 2.4.1 beschrieben ist, für jedes Dokument einzeln berechnet. Außerdem muss auch die erwartende Ergebnisliste von 10 relevantesten Dokumenten festgelegt und eingestuft werden.

#	Arbeitsbezeichnung	Datum
1	Senior Java-Entwickler Angebotsimport	01.10.2015
2	SOFTWAREENTWICKLER	07.10.2015
3	Senior Software Developer (Java/Backend)	21.09.2015
4	Senior Software Developer (Java) – Single Sign-On	21.09.2015
5	IT-Consultant ERP/ Softwareentwickler	07.10.2015
6	Softwareentwickler/in - Berlin	01.10.2015
7	Senior Softwareentwickler/-architekt Java EE	01.10.2015
8	Web-Entwickler	01.10.2015
9	Softwareentwickler Java für Versicherungen	30.09.2015
10	Webentwickler Frontend	27.09.2015

Tabelle 5.7: Die erwartende Ergebnismenge für Volltextsuche

Um die Testmenge zu bestimmen, werden 3 booleschen Suchanfragen ausgeführt und nach Datum sortiert ausgegeben. Die Arbeitsangebote „Senior Java-Entwickler Angebotsimport“, „Senior Software Developer (Java/Backend)“ und „Senior Software Developer (Java) – Single Sign-On“ beinhalten sowohl „Softwareentwickler Java“ als auch „Testautomatisierung“ Terme. Das Arbeitsangebot „Webentwickler Frontend“ ist das einzige Dokument, das nur den Term „Testautomatisierung“ enthält. Die restliche Arbeitsangebote sind die Ergebnisse für die Suchanfrage „Softwareentwickler Java“. Bei der Festlegung von Position innerhalb der erwartenden Ergebnisliste wurde auf die Anzahl von gesuchten Termen und das Datum geachtet. Unter Berücksichtigung der neuen Testmenge können *Recall* und *Precision* für 10 ersten Hits der ersten *Query* berechnet werden.

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	Softwareentwickler RefactoringInterhyp AG	15.07.2015	0	0	0
2	Softwareentwickler/ -architekt	09.07.2015	0	0	0
3	Senior Java-Entwickler Angebotsimport	01.10.2015	1	0.1	0.33
4	Senior Java-Entwickler Full Stack	14.05.2015	1	0.1	0.25
5	Senior Software Developer (Java/-Backend)	21.09.2015	2	0.2	0.4
6	Senior Software Developer (Java) – Single Sign-On	21.09.2015	3	0.3	0.5
7	Spezialist Testautomatisierung	10.07.2015	3	0.3	0.43
8	Softwareentwickler	18.05.2015	3	0.3	0.38
9	Java-Softwareentwickler	08.05.2015	3	0.3	0.33
10	Java Softwareentwickler	21.05.2015	3	0.3	0.3

Tabelle 5.8: Query Nr. 1.2 - 10 beste Hits

Die zwei erste Hits in der Ergebnisliste gehören nicht zu den erwartenden Arbeitsangeboten und ihre *Recall* und *Precision* sind deswegen 0. Das Arbeitsangebot „Senior Java-Entwickler Angebotsimport“ wird erwartet und das ist auch das erste gefundene Dokument.

$$Recall = \frac{relevanteDokumenteBisher}{AnzahlErwartendenDokumenten} \Rightarrow \frac{1}{10} = 0.1$$

$$Precision = \frac{relevanteDokumenteBisher}{AnzahlDokumentenBisher} \Rightarrow \frac{1}{3} = 0.33$$

Das nächste Arbeitsangebot wird wieder nicht erwartet. Deswegen bleibt auch seine *Recall* unverändert. Der *Precision* Wert wird kleiner $1/4 = 0.25$. Die restliche *Recall* und *Precision* Werte werden analog zu dem Beispiel berechnet. Nachdem alle *Recall* und *Precision* Werte bekannt wurden, kann auch *interpolated precision* für jeden *Recall*-Punkt berechnet wurden.

Recall	0.1	0.2	0.3
interpolated precision	0.33	0.4	0.5

Tabelle 5.9: Query Nr. 1.2 - Interpolated precision

Query Nr. 2

```
{
  "bool": {
    "should": [
      {
        "match": {
          "content": {
            "query": "Softwareentwickler Java",
            "type": "boolean",
            "operator": "AND"
          }
        }
      },
      {
        "match": {
          "content": {
            "query": "Testautomatisierung",
            "type": "boolean"
          }
        }
      }
    ],
    "minimum_should_match": "1"
  },
  "sort": [ { "receivedDate": { "order": "desc" } } ]
}
```

Beschreibung: Die zweite *Query* sortiert einfach die Ergebnisliste absteigend nach dem Feld *receivedDate*, das als Datum repräsentiert wird.

Ergebnis: Die Tabelle(5.10) zeigt, dass die Ergebnismenge jetzt absteigend nach Datum sortiert ist. Die Folge von *Recall* und *Interpolated precision* sind jetzt im Vergleich zu der alten Suchanfrage viel besser geworden.

Interessant ist auch der *Explain*-Plan von dieser *Query*, dessen Ausschnitt im Anhang (siehe Anhang 5.1: Query Nr. 2 - Explain-Plan des ersten Hits) zu finden ist. Die eigentliche *Score*-Berechnung für den ersten Hit ist **0.12759583** wird aber durch **NaN** ersetzt. **NaN** steht in *ElasticSearch* für maximale *Score*. Das heißt, dass die Dokumente, die eigentlich für die Suchanfrage nicht wirklich relevant sein könnten, durch Sortiervorgang und das Maximale *Score* jedoch relevant werden. Daraus folgt, dass die *Score* bei der Sortierung nicht mehr berücksichtigt wird.

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	SOFTWAREENTWICKLER	07.10.2015	1	0.1	1
2	IT-Consultant ERP/ Softwareentwickler	07.10.2015	2	0.2	1
3	Softwareentwickler/in - Berlin	01.10.2015	3	0.3	1
4	Senior Softwareentwickler/-architekt Java EE	01.10.2015	4	0.4	1
5	Web-Entwickler	01.10.2015	5	0.5	1
6	Senior Java-Entwickler Angebotssimport	01.10.2015	6	0.6	1
7	Softwareentwickler Java für Versicherungen	30.09.2015	7	0.7	1
8	Softwareentwickler (Java) für Banken	30.09.2015	7	0.7	0.88
9	Hochschulabsolvent - Softwareentwickler Java	30.09.2015	7	0.7	0.78
10	Softwareentwickler Lotus Notes	29.09.2015	7	0.7	0.7

Tabelle 5.10: Query Nr. 2 - 10 beste Hits

Recall	0.1	0.2	0.3	0.4	0.5	0.6	0.7
interpolated precision	1	1	1	1	1	1	1

Tabelle 5.11: Query Nr. 2 - Interpolated precision

Query Nr. 3

```
{
  "function_score": {
    "query": {
      "bool": {
        "should": [
          {
            "match": {
              "content": {
                "query": "Softwareentwickler Java",
                "type": "boolean",
                "operator": "AND"
              }
            }
          },
          {
            "match": {
              "content": {
                "query": "Testautomatisierung",

```

```

        "type": "boolean",
        "operator": "AND"
      }
    }}
  ]]
},
"functions": [
  {
    "exp": {
      "receivedDate": {
        "scale": "14d",
        "decay": 0.5
      }
    }
  }
]
}

```

Beschreibung: Die dritte *Query* ersetzt die Sortierung durch die *Decay*-Funktion (siehe Kapitel 2.5), um das Datum bei der *Score*-Berechnung zu berücksichtigen. Das Datum-Score wird jetzt *exponential* mit dem festgelegten *Scale*-Wert von 14 Tagen berechnet.

Ergebnis:

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
2	Senior Software Developer (Java/-Backend)	21.09.2015	2	0.2	1
3	Senior Software Developer (Java) – Single Sign-On	21.09.2015	3	0.3	1
4	Softwareentwickler/in - Berlin	01.10.2015	4	0.4	1
5	Senior Softwareentwickler/-architekt Java EE	01.10.2015	5	0.5	1
6	Softwareentwickler Java für Versicherungen	30.09.2015	6	0.6	1
7	SOFTWAREENTWICKLER	07.10.2015	7	0.7	1
8	Hochschulabsolvent - Softwareentwickler Java	30.09.2015	7	0.7	0.88
9	Softwareentwickler (Java) für Banken	30.09.2015	7	0.7	0.78
10	Java Experte/Softwareentwickler	24.09.2015	7	0.7	0.7

Tabelle 5.12: Query Nr. 3 - 10 beste Hits

Der *Explain*-Plan im Anhang(Anhang 5.2: Query Nr. 3 - Explain-Plan des ersten 3 Hits) zeigt schon deutlich, dass das Datum bei der *Score* mitgerechnet wird. Mit der aktuellen Testmenge sind *Recall* und *Interpolated precision* unverändert geblieben. Die Reihenfolge von Dokumenten entspricht jetzt aber mehr den Erwartungen.

Recall	0.1	0.2	0.3	0.4	0.5	0.6	0.7
interpolated precision	1	1	1	1	1	1	1

Tabelle 5.13: Query Nr. 3 - Interpolated precision

Query Nr. 4

```
{
  "function_score": {
    "query": {
      "bool": {
        "must": {
          "term": {
            "read": false
          }
        },
        "should": [
          {
            "match": {
              "content": {
                "query": "Softwareentwickler Java",
                "type": "boolean",
                "operator": "AND"
              }
            }
          },
          {
            "match": {
              "content": {
                "query": "Testautomatisierung",
                "type": "boolean",
                "operator": "AND"
              }
            }
          }
        ]
      }
    },
    "functions": [
      {
        "exp": {
          "receivedDate": {
            "scale": "14d",
            "decay": 0.5
          }
        }
      }
    ]
  }
}
```

Beschreibung: Die *Query* Nr. 4 erweitert die letzte *Query* mit einer neuen booleschen *Term-Query*, die dafür sorgt, dass nur ungelesene Arbeitsangebote durchgesucht werden.

Ergebnis: Interessanterweise hat diese winzige Änderung eine neue Ergebnisliste verursacht, was nicht erwartet und auch nie gewünscht wurde.

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
2	SOFTWAREENTWICKLER	07.10.2015	2	0.2	1
3	Softwareentwickler/in - Berlin	01.10.2015	3	0.3	1
4	Senior Softwareentwickler/-architekt Java EE	01.10.2015	4	0.4	1
5	Senior Software Developer (Java/-Backend)	21.09.2015	5	0.5	1
6	Senior Software Developer (Java) – Single Sign-On	21.09.2015	6	0.6	1
7	Softwareentwickler Java für Versicherungen	30.09.2015	7	0.7	1
8	Hochschulabsolvent - Softwareentwickler Java	30.09.2015	7	0.7	0.88
9	IT-Consultant ERP/ Softwareentwickler	07.10.2015	8	0.8	0.89
10	Softwareentwickler (Java) für Banken	30.09.2015	8	0.8	0.8

Tabelle 5.14: Query Nr. 4 - 10 beste Hits

Recall	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
interpolated precision	1	1	1	1	1	1	1	0.89

Tabelle 5.15: Query Nr. 4 - Interpolated precision

Aus der Tabelle oben folgt, dass die neue Ergebnismenge im Vergleich zu dem alten Ergebnis besser geworden ist. Die Untersuchung von *Explain-Plan* (Anhang 5.3: Query Nr. 4 - *Explain-Plan* des ersten Hit) hat gezeigt, dass die neue boolesche *Term-Query* bei der *Score*-Berechnung mitgezählt wird. Das darf aber nicht passieren. Das Feld *read* ist ein Flag, der ein Dokument aus dem Suchvorgang komplett ausschliessen soll und keinen Einfluss auf seine Relevanz haben darf.

Query Nr. 5

```
{
  "function_score" : {
    "query" : {
      "filtered" : {
        "query" : {
          "bool" : {
            "should" : [ {
              "match" : {
                "content" : {
                  "query" : "Softwareentwickler Java",
                  "type" : "boolean",
                  "operator" : "AND"
                }
              }
            }, {
              "match" : {
                "content" : {
                  "query" : "Testautomatisierung",
                  "type" : "boolean",
                  "operator" : "AND"
                }
              }
            }
          ]
        }
      },
      "filter" : {
        "term" : {
          "read" : false
        }
      }
    },
    "functions" : [ {
      "exp" : {
        "receivedDate" : {
          "scale" : "14d"
        }
      }
    } ]
  }
}
```

Beschreibung: Die fünfte und auch die letzte *Query* im Zusammenhang mit der Volltextsuche verwendet einen Filter statt eine boolesche Term-Anfrage, um das Statusfeld *read* anzufragen. Der Filter kann jetzt die Dokumente nach dem *Read*-Status ohne Einfluss auf ihre Relevanz filtern.

Ergebnis: Die Ergebnismenge wird hier nicht extra aufgelistet, weil sie äquivalent zu dem Ergebnis der dritten *Query* ist. Der *Explain*-Plan des ersten Hits befindet sich im Anhang (Anhang 5.4: Query Nr. 5 - Explain-Plan des ersten Hits).

5.2 Gewichtete Suchanfragen

Die gewichtete Suchanfrage verwendet als Suchterme die persönlichen Fähigkeiten und Qualifikationen, die der Benutzer selbst eingeschätzt hat. Diese Einschätzung muss jedenfalls bei der Suche berücksichtigt werden. Damit es sinnvoll getestet werden könnte, muss die Suchanfrage aus mehreren Suchbegriffen mit unterschiedlicher Einschätzung bestehen. Die *Query*-Struktur wird dadurch größer und wird deswegen in den Anhang ausgelagert.

Die Testszenarien beinhaltet vier unterschiedlich geschätzten Suchbegriffe.

Qualifikation	Einschätzung
Spring Framework	5
Java	3
PostgreSQL	2
Datenbanken	2

Tabelle 5.16: Einschätzung von Qualifikationen für die Evaluierung

Die Tabelle(5.17) zeigt auch die Anzahl von relevanten Dokumenten für die oben aufgelistete Suchbegriffe.

Suchbegriff	relevante Dokumente
Spring	151
Framework	136
Spring {AND} Framework	43
Java	305
PostgreSQL	17
Datenbanken	171
Spring {AND} Framework {AND} Java {AND} PostgreSQL {AND} Datenbanken	2

Tabelle 5.17: Relevanz von Suchbegriffen für die gewichtete Suche

Die erwartende Ergebnismenge wird in der Tabelle(5.18) zusammengefasst. Bei der Relevanzbestimmung wird auf das Vorkommen von Suchbegriffen in Abhängigkeit von ihrer Schätzung geachtet.

#	Arbeitsbezeichnung	Datum
1	Java Software Developer Connected Mobility	01.10.2015
2	Senior Java-Entwickler Angebotsimport	01.10.2015
3	Softwareentwickler (Java) für Banken	30.09.2015
4	Softwareentwickler Java für Versicherungen	30.09.2015
5	Java Developer! Inhouse Java Software Entwickler	28.09.2015
6	Software Developer (Java/Backend)	21.09.2015
7	Software Developer DWH	27.09.2015
8	Java Developer JEE / J2EE	03.10.2015
9	Junior Datenbank Analyst-Bereich Reise	02.10.2015
10	Softwareentwickler (Java) für Banken	30.09.2015

Tabelle 5.18: Die erwartende Ergebnismenge für die gewichtete Suche

Query Nr. 6

Beschreibung: Die 6. Query (Anhang 5.5: Query Nr. 6) benutzt die *Boost*-Funktion, um die Score-Wert für einen bestimmten Suchterm mit dem *Boost factor* zu multiplizieren. Die *Boost factor* wird nach der Formel $Factor = Einschätzung * 10$ berechnet.

Ergebnis: Die Query findet nur fünf relevante Dokumente. Da die relevanteste Dokumente jedoch gefunden werden, wird die Gesamtrelevanz der Ergebnismenge trotzdem positiv bewertet. Aus dem *Explain*-Plan (Anhang 5.6: Query Nr. 6 - Explain-Plan des ersten Hit) wird es ersichtlich, dass nur ein exakter Suchtreffer geboostet wird. So werden nur die Begriffe, *Java* und *Datenbank*, aber nicht *Spring Framework* geboostet.

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
2	Softwareentwickler Java für Versi-cherungen	30.09.2015	2	0.2	1
3	Software Developer DWH	27.09.2015	3	0.3	1
4	Java Software Developer Connec-ted Mobility	01.10.2015	4	0.4	1
5	Java Developer! Inhouse Java Software Entwickler	28.09.2015	5	0.5	1
6	Software Developer (PHP / Full Stack)	01.10.2015	5	0.5	0.83
7	Java Backend Entwickler	28.09.2015	5	0.5	0.71
8	Junior Softwareentwickler/in Java	07.09.2015	5	0.5	0.63
9	Senior Software Developer (Java)	21.09.2015	5	0.5	0.56
10	Java Frontend Developer	19.09.2015	5	0.5	0.5

Tabelle 5.19: Query Nr. 6 - 10 beste Hits

Recall	0.1	0.2	0.3	0.4	0.5
interpolated precision	1	1	1	1	1

Tabelle 5.20: Query Nr. 6 - Interpolated precision

Query Nr. 7

Beschreibung: Die letzte *Query* (Anhang 5.7: Query Nr. 7) verknüpft alle Suchterme mit *OR* und verwendet eine zusätzliche *Weight*-Funktion, um die Score noch einmal zu boosten. Im Zusammenhang mit *OR*-Verknüpfung werden jetzt alle Dokumente mit dem *Spring*- oder *Framework*-Term höher gewichtet.

Ergebnis: Obwohl die Ergebnisliste sich für 10 ersten Hits kaum geändert hat, ist die *Query* insgesamt besser geworden. So wird beispielsweise die Position des Arbeitsangebotes „Softwareentwickler (Java) für Banken“ um 10 Stellen (von 19 zu 9) geändert. Die *Query* hat allerdings auch einen Nachteil und zwar die Arbeitsangebote, wo das Wort *Framework* nicht im Zusammenhang mit *Spring* sondern mit *Play* oder *AngularJS* vorkommt, ebenfalls geboosted werden.

#	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
2	Softwareentwickler Java für Versi-cherungen	30.09.2015	2	0.2	1
3	Software Developer DWH	27.09.2015	3	0.3	1
4	Java Software Developer Connec-ted Mobility	01.10.2015	4	0.4	1
5	Java Developer! Inhouse Java Software Entwickler	28.09.2015	5	0.5	1
6	Software Developer (PHP / Full Stack)	01.10.2015	5	0.5	0.83
7	Java Backend Entwickler	28.09.2015	5	0.5	0.71
8	Senior Software Developer (Java)	21.09.2015	5	0.5	0.63
9	Softwareentwickler (Java) für Ban-ken	30.09.2015	6	0.6	0.68
10	Junior Softwareentwickler/in Java	07.09.2015	6	0.6	0.6

Tabelle 5.21: Query Nr. 7 - 10 beste Hits mit der Relevanzeinstufung

Recall	0.1	0.2	0.3	0.4	0.5	0.6
interpolated precision	1	1	1	1	1	0.68

Tabelle 5.22: Query Nr. 7 - Interpolated precision

5.3 Vergleich und Fazit

Um Volltextsuche mit der gewichteten Suche sinnvoll vergleichen zu können, werden die *Query* Nr.5 und die *Query* Nr. 7 mit den gleichen Suchtermen(siehe Tabelle 5.16) ausgeführt. Aus der Tabelle(5.23) kann man erschließen, dass die Werte für *Recall* und *Interpolated precision* der Volltext-Suchanfrage äquivalent zu den Werten der 6. Query sind. Die Untersuchung von Arbeitsangeboten „*Senior Software Developer (Java)*“ und „*Softwareentwickler (Java) für Banken*“ aus der gewichteten Suchanfrage hat ergeben, dass diese Arbeitsangebote aus der Benutzersicht viel relevanter als die Arbeitsangebote aus der Volltextsuche („*Senior) PHP-Entwickler*“, „*Software Engineer Backend Zalando*“) sind. Wenn man die *Recall*, *Interpolated precision* und Benutzerbewertung zusammenfasst, kommt man zu dem Ergebnis, dass die gewichte Suche im Vergleich zur Volltextsuche insgesamt mehr relevante Dokumente liefert.

#	Query	Arbeitsbezeichnung	Datum	Rel. Dok. bisher	Rec.	Prec.
1	5	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
1	7	Senior Java-Entwickler Angebot-simport	01.10.2015	1	0.1	1
2	5	Softwareentwickler Java für Versi-cherungen	30.09.2015	2	0.2	1
2	7	Softwareentwickler Java für Versi-cherungen	30.09.2015	2	0.2	1
3	5	Software Developer DWH	27.09.2015	3	0.3	1
3	7	Software Developer DWH	27.09.2015	3	0.3	1
4	5	Java Software Developer Connec-ted Mobility	01.10.2015	4	0.4	1
4	7	Java Software Developer Connec-ted Mobility	01.10.2015	4	0.4	1
5	5	Java Developer! Inhouse Java Software Entwickler	28.09.2015	5	0.5	1
5	7	Java Developer! Inhouse Java Software Entwickler	28.09.2015	5	0.5	1
6	5	Software Developer (PHP / Full Stack)	01.10.2015	5	0.5	0.83
6	7	Software Developer (PHP / Full Stack)	01.10.2015	5	0.5	0.83
7	5	Java Backend Entwickler	28.09.2015	5	0.5	0.71
7	7	Java Backend Entwickler	28.09.2015	5	0.5	0.71
8	5	(Senior) PHP-Entwickler	21.09.2015	5	0.5	0.63
8	7	Senior Software Developer (Java)	21.09.2015	5	0.5	0.63
9	5	Software Engineer Backend Zalan-do	30.09.2015	5	0.5	0.56
9	7	Softwareentwickler (Java) für Ban-ken	30.09.2015	6	0.6	0.68
10	5	Junior Softwareentwickler/in Java	07.09.2015	5	0.5	0.5
10	7	Junior Softwareentwickler/in Java	07.09.2015	6	0.6	0.6

Tabelle 5.23: Vergleich Query Nr. 5 und Query Nr. 7

6. Problemen und Schwierigkeiten

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7. Zusammenfassung und Ausblick

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Literaturverzeichnis

- Dorschel, Joachim (2015). „Praxishandbuch Big Data - Wirtschaft – Recht – Technik“. 1. Aufl. Berlin Heidelberg New York: Springer-Verlag (siehe S. 4, 6).
- Jan, Steemann (2013). *Datenmodellierung in nicht relationalen Datenbanken*. url: <https://entwickler.de/online/datenbanken/datenmodellierung-in-nicht-relationalen-datenbanken-137872.html> (siehe S. 6).
- OnPage (2016). *Tag Cloud*. url: https://de.onpage.org/wiki/Tag_Cloud (siehe S. 12).
- Henrich, Andreas (2008). „Information Retrieval Grundlagen, Modelle und Anwendungen“. 1. Aufl. Otto-Friedrich-Universität Bamberg (siehe S. 12 f.).
- Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). „Introduction to Information Retrieval“. Anniversary. Cambridge: Cambridge University Press (siehe S. 14).
- Mandl, Thomas (2001). „Tolerantes Information Retrieval“. Hochschulverband für Informatikwissenschaft(HI)e.V (siehe S. 15).
- Salton, G.; Wong, A.; Yang, C. S. (1975). „A Vector Space Model for Automatic Indexing“. *Commun. ACM* 18.11, S. 613–620. doi: 10.1145/361219.361220. url: <http://doi.acm.org/10.1145/361219.361220> (siehe S. 16).
- Rafal Kuc, Marek Rogozinski (2013). „Mastering Elasticsearch“. 1. Aufl. Packt Publishing Ltd. (siehe S. 20).
- Terrier, Francois (2013). *On Elasticsearch performance*. url: <https://blog.liip.ch/archive/2013/07/19/on-elasticsearch-performance.html> (siehe S. 21).

Abbildungsverzeichnis

2.1 Relationale Tabellen <i>skill-categories</i> und <i>skills</i>	4
2.2 Prozessablauf in IntelliJob	8
2.3 Schichtenarchitektur	10
2.4 Tag Clouds - <i>www.einfachbewusst.de</i>	11
2.5 Prozess des Wissenstransfers vom Autor eines Dokumentes bis zum Informations-Nachfragenden	13
2.6 Die elementaren booleschen Operationen	15
2.7 Vektorraumpräsentation eines Dokumentraums (Salton u. McGill 1987, 129)	16
2.8 Visualisierung von <i>Decay</i> -Funktion mit <i>Gaus</i>	20
2.9 Beispiel - Ein Cluster mit zwei Knoten, 4 <i>shards</i> und 1 <i>replica shard</i> [Terrier 2013]	21

Tabellenverzeichnis

2.1 Begriffsvergleich zwischen Relationale Datenbanken und ElasticSearch	18
5.1 Datenausgangszustand für die Evaluierung	31
5.2 Query Nr. 1 - Ergebnis-Ausschnitt	32
5.3 Query Nr. 1.1 - Ergebnis-Ausschnitt	33
5.4 Relevanz von Suchbegriffen für die Volltextsuche	33
5.5 Query Nr. 1.2 - Ergebnis- 5 beste Hits	34
5.6 Query Nr. 1.1 - falsch-positive Recall, Precision	34
5.7 Die erwartende Ergebnismenge für Volltextsuche	35
5.8 Query Nr. 1.2 - 10 beste Hits	36
5.9 Query Nr. 1.2 - Interpolated precision	36
5.10Query Nr. 2 - 10 beste Hits	38
5.11Query Nr. 2 - Interpolated precision	38
5.12Query Nr. 3 - 10 beste Hits	39
5.13Query Nr. 3 - Interpolated precision	40
5.14Query Nr. 4 - 10 beste Hits	41
5.15Query Nr. 4 - Interpolated precision	41
5.16Einschätzung von Qualifikationen für die Evaluierung	43
5.17Relevanz von Suchbegriffen für die gewichtete Suche	43
5.18Die erwartende Ergebnismenge für die gewichtete Suche	44
5.19Query Nr. 6 - 10 beste Hits	45
5.20Query Nr. 6 - Interpolated precision	45
5.21Query Nr. 7 - 10 beste Hits mit der Relevanzeinstufung	46
5.22Query Nr. 7 - Interpolated precision	46
5.23Vergleich Query Nr. 5 und Query Nr. 7	47

Glossar

Chunker - Machine Learning Technologie - Teilt Text in syntaktisch korrelierten Teile von Wörtern, wie Nomen Gruppen.

Coreference Resolution - Machine Learning Technologie - Bezugnahme auf dieselbe Entität.

DAO - Data Access Object - Adapter zur Abstrahierung und Entkopplung von Datenzugriffe.

Invertierter Index - wird in Suchserver eingesetzt und bezieht sich auf die Tatsache, dass nicht die Dokumente auf Worte zeigen, sondern eine Liste von Worten auf Dokumente zeigt.

IR - Information Retrieval

Named Entity Recognition - Machine Learning Technologie - erkennt und klassifiziert Bestandteile im Text.

NoSQL - not only SQL - bezeichnet Datenbanken, die einen nicht-relationalen Ansatz verfolgen.

Part-Of-Speech tagging - Machine Learning Technologie - Das Zuweisen von Markierungen zu einzelnen Einheiten (Wortart-Annotierung).

SentenceDetector - Machine Learning Technologie - teilt Text in einzelne Sätze

Tokenizer - Machine Learning Technologie - teilt Sätze in einzelne Worte

Anhang

Anhang 5.1: Query Nr. 2 - Explain-Plan des ersten Hits

```
1 Hit: score(NaN) | receivedDate(07.10.2015 11-24) |
      name(SOFTWAREENTWICKLER (M/W)) | id(5672646344aeeb036a01be0a)
1 Explain(: 0.12759583 = (MATCH) product of:
  0.25519165 = (MATCH) sum of:
    0.25519165 = (MATCH) sum of:
      0.19744588 = (MATCH) weight(content:softwareentwickl in 364)
        [PerFieldSimilarity], result of:
          0.19744588 = score(doc=364,freq=5.0), product of:
            0.50549334 = queryWeight, product of:
              3.7265449 = idf(docFreq=149, maxDocs=2292)
              0.13564666 = queryNorm
            0.39060035 = fieldWeight in 364, product of:
              2.236068 = tf(freq=5.0), with freq of:
                5.0 = termFreq=5.0
              3.7265449 = idf(docFreq=149, maxDocs=2292)
              0.046875 = fieldNorm(doc=364)
          0.057745762 = (MATCH) weight(content:java in 364)
            [PerFieldSimilarity], result of:
              0.057745762 = score(doc=364,freq=1.0), product of:
                0.40878406 = queryWeight, product of:
                  3.0135949 = idf(docFreq=305, maxDocs=2292)
                  0.13564666 = queryNorm
                0.14126226 = fieldWeight in 364, product of:
                  1.0 = tf(freq=1.0), with freq of:
                    1.0 = termFreq=1.0
                  3.0135949 = idf(docFreq=305, maxDocs=2292)
                  0.046875 = fieldNorm(doc=364)
      0.5 = coord(1/2)
    )
)
```

Anhang 5.2: Query Nr. 3 - Explain-Plan des ersten 3 Hits

```

1 Hit: score(4.9161434E-4) | receivedDate(01.10.2015 02-55) | name(Senior
    Java-Entwickler Angebotsimport (m/w)Idealo Internet GmbH - Berlin) |
    id(567264c744aeeb036a01be31)
1 Explain(: 4.916144E-4 = (MATCH) function score, product of:
    0.431067 = (MATCH) sum of:
        0.19829285 = (MATCH) sum of:
            0.103017226 = (MATCH) weight(content:softwareentwickl in 403)
                [PerFieldSimilarity], result of:
                    0.103017226 = score(doc=403,freq=1.0), product of:
                        0.50549334 = queryWeight, product of:
                            3.7265449 = idf(docFreq=149, maxDocs=2292)
                            0.13564666 = queryNorm
                        0.20379542 = fieldWeight in 403, product of:
                            1.0 = tf(freq=1.0), with freq of:
                                1.0 = termFreq=1.0
                                3.7265449 = idf(docFreq=149, maxDocs=2292)
                                0.0546875 = fieldNorm(doc=403)
                            0.09527563 = (MATCH) weight(content:java in 403) [PerFieldSimilarity],
                                result of:
                                    0.09527563 = score(doc=403,freq=2.0), product of:
                                        0.40878406 = queryWeight, product of:
                                            3.0135949 = idf(docFreq=305, maxDocs=2292)
                                            0.13564666 = queryNorm
                                        0.23307082 = fieldWeight in 403, product of:
                                            1.4142135 = tf(freq=2.0), with freq of:
                                                2.0 = termFreq=2.0
                                                3.0135949 = idf(docFreq=305, maxDocs=2292)
                                                0.0546875 = fieldNorm(doc=403)
                                    0.23277412 = (MATCH) weight(content:testautomatisierung in 403)
                                        [PerFieldSimilarity], result of:
                                            0.23277412 = score(doc=403,freq=1.0), product of:
                                                0.75984997 = queryWeight, product of:
                                                    5.601686 = idf(docFreq=22, maxDocs=2292)
                                                    0.13564666 = queryNorm
                                                0.3063422 = fieldWeight in 403, product of:
                                                    1.0 = tf(freq=1.0), with freq of:
                                                        1.0 = termFreq=1.0
                                                        5.601686 = idf(docFreq=22, maxDocs=2292)
                                                        0.0546875 = fieldNorm(doc=403)
                                            0.0011404593 = (MATCH) Math.min of
                                                0.0011404593 = (MATCH) Function for field receivedDate:
                                                    0.0011404593 = exp(- MIN[Math.max(Math.abs(1.443660906E12(=doc value)
                                                        - 1.455486160446E12(=origin))) - 0.0(=offset), 0]) *
                                                        5.73038343716886E-10)
                                                    3.4028235E38 = maxBoost
                                                1.0 = queryBoost
                                )

```

TABELLENVERZEICHNIS

```
-----
2 Hit: score(2.8052164E-4) | receivedDate(21.09.2015 03-23) | name(Senior
    Software Developer (Java / Backend) - User Management (m/w)Idealo
    Internet GmbH - Berlin) | id(56726eb344aeeb036a01c165)
2 Explain(: 2.8052164E-4 = (MATCH) function score, product of:
0.4031614 = (MATCH) sum of:
0.17038727 = (MATCH) sum of:
0.103017226 = (MATCH) weight(content:softwareentwickl in 1222)
    [PerFieldSimilarity], result of:
0.103017226 = score(doc=1222,freq=1.0), product of:
0.50549334 = queryWeight, product of:
    3.7265449 = idf(docFreq=149, maxDocs=2292)
    0.13564666 = queryNorm
0.20379542 = fieldWeight in 1222, product of:
    1.0 = tf(freq=1.0), with freq of:
        1.0 = termFreq=1.0
        3.7265449 = idf(docFreq=149, maxDocs=2292)
        0.0546875 = fieldNorm(doc=1222)
0.06737005 = (MATCH) weight(content:java in 1222)
    [PerFieldSimilarity], result of:
0.06737005 = score(doc=1222,freq=1.0), product of:
0.40878406 = queryWeight, product of:
    3.0135949 = idf(docFreq=305, maxDocs=2292)
    0.13564666 = queryNorm
0.16480596 = fieldWeight in 1222, product of:
    1.0 = tf(freq=1.0), with freq of:
        1.0 = termFreq=1.0
        3.0135949 = idf(docFreq=305, maxDocs=2292)
        0.0546875 = fieldNorm(doc=1222)
0.23277412 = (MATCH) weight(content:testautomatisierung in 1222)
    [PerFieldSimilarity], result of:
0.23277412 = score(doc=1222,freq=1.0), product of:
0.75984997 = queryWeight, product of:
    5.601686 = idf(docFreq=22, maxDocs=2292)
    0.13564666 = queryNorm
0.3063422 = fieldWeight in 1222, product of:
    1.0 = tf(freq=1.0), with freq of:
        1.0 = termFreq=1.0
        5.601686 = idf(docFreq=22, maxDocs=2292)
        0.0546875 = fieldNorm(doc=1222)
6.958048E-4 = (MATCH) Math.min of
6.958048E-4 = (MATCH) Function for field receivedDate:
6.958048E-4 = exp(- MIN[Math.max(Math.abs(1.44279863E12(=doc value) -
    1.455486160446E12(=origin))) - 0.0(=offset), 0)] *
    5.73038343716886E-10)
3.4028235E38 = maxBoost
1.0 = queryBoost
)
-----
```

TABELLENVERZEICHNIS

```
3 Hit: score(2.8052164E-4) | receivedDate(21.09.2015 03-23) | name(Senior
    Software Developer (Java) Single Sign-On) | id(567276e244aeeb036a01c3d0)
3 Explain(: 2.8052164E-4 = (MATCH) function score, product of:
    0.4031614 = (MATCH) sum of:
        0.17038727 = (MATCH) sum of:
            0.103017226 = (MATCH) weight(content:softwareentwickl in 1840)
                [PerFieldSimilarity], result of:
                    0.103017226 = score(doc=1840,freq=1.0), product of:
                        0.50549334 = queryWeight, product of:
                            3.7265449 = idf(docFreq=149, maxDocs=2292)
                            0.13564666 = queryNorm
                        0.20379542 = fieldWeight in 1840, product of:
                            1.0 = tf(freq=1.0), with freq of:
                                1.0 = termFreq=1.0
                                3.7265449 = idf(docFreq=149, maxDocs=2292)
                                0.0546875 = fieldNorm(doc=1840)
                    0.06737005 = (MATCH) weight(content:java in 1840)
                        [PerFieldSimilarity], result of:
                            0.06737005 = score(doc=1840,freq=1.0), product of:
                                0.40878406 = queryWeight, product of:
                                    3.0135949 = idf(docFreq=305, maxDocs=2292)
                                    0.13564666 = queryNorm
                                0.16480596 = fieldWeight in 1840, product of:
                                    1.0 = tf(freq=1.0), with freq of:
                                        1.0 = termFreq=1.0
                                        3.0135949 = idf(docFreq=305, maxDocs=2292)
                                        0.0546875 = fieldNorm(doc=1840)
                            0.23277412 = (MATCH) weight(content:testautomatisierung in 1840)
                                [PerFieldSimilarity], result of:
                                    0.23277412 = score(doc=1840,freq=1.0), product of:
                                        0.75984997 = queryWeight, product of:
                                            5.601686 = idf(docFreq=22, maxDocs=2292)
                                            0.13564666 = queryNorm
                                        0.3063422 = fieldWeight in 1840, product of:
                                            1.0 = tf(freq=1.0), with freq of:
                                                1.0 = termFreq=1.0
                                                5.601686 = idf(docFreq=22, maxDocs=2292)
                                                0.0546875 = fieldNorm(doc=1840)
        6.958048E-4 = (MATCH) Math.min of
            6.958048E-4 = (MATCH) Function for field receivedDate:
                6.958048E-4 = exp(- MIN[Math.max(Math.abs(1.44279863E12(=doc value) -
                    1.455486160446E12(=origin))) - 0.0(=offset), 0)] *
                    5.73038343716886E-10)
            3.4028235E38 = maxBoost
    1.0 = queryBoost
)
```

Anhang 5.3: Query Nr. 4 - Explain-Plan des ersten Hit

```

1 Hit: score(6.45385E-4) | receivedDate(01.10.2015 02-55) | name(Senior
      Java-Entwickler Angebotsimport (m/w)Idealo Internet GmbH - Berlin) |
      id(567264c744aeeb036a01be31)
1 Explain(: 6.45385E-4 = (MATCH) function score, product of:
  0.56651473 = (MATCH) sum of:
    0.13950714 = (MATCH) weight(read:F in 403) [PerFieldSimilarity], result
      of:
      0.13950714 = score(doc=403,freq=1.0), product of:
        0.1369141 = queryWeight, product of:
          1.0189391 = idf(docFreq=2248, maxDocs=2292)
          0.13436927 = queryNorm
        1.0189391 = fieldWeight in 403, product of:
          1.0 = tf(freq=1.0), with freq of:
            1.0 = termFreq=1.0
          1.0189391 = idf(docFreq=2248, maxDocs=2292)
          1.0 = fieldNorm(doc=403)
      0.19642553 = (MATCH) sum of:
        0.102047116 = (MATCH) weight(content:softwareentwickl in 403)
          [PerFieldSimilarity], result of:
          0.102047116 = score(doc=403,freq=1.0), product of:
            0.50073314 = queryWeight, product of:
              3.7265449 = idf(docFreq=149, maxDocs=2292)
              0.13436927 = queryNorm
            0.20379542 = fieldWeight in 403, product of:
              1.0 = tf(freq=1.0), with freq of:
                1.0 = termFreq=1.0
              3.7265449 = idf(docFreq=149, maxDocs=2292)
              0.0546875 = fieldNorm(doc=403)
          0.09437842 = (MATCH) weight(content:java in 403) [PerFieldSimilarity],
            result of:
            0.09437842 = score(doc=403,freq=2.0), product of:
              0.40493453 = queryWeight, product of:
                3.0135949 = idf(docFreq=305, maxDocs=2292)
                0.13436927 = queryNorm
              0.23307082 = fieldWeight in 403, product of:
                1.4142135 = tf(freq=2.0), with freq of:
                  2.0 = termFreq=2.0
                3.0135949 = idf(docFreq=305, maxDocs=2292)
                0.0546875 = fieldNorm(doc=403)
        0.23058207 = (MATCH) weight(content:testautomatisierung in 403)
          [PerFieldSimilarity], result of:
          0.23058207 = score(doc=403,freq=1.0), product of:
            0.7526944 = queryWeight, product of:
              5.601686 = idf(docFreq=22, maxDocs=2292)
              0.13436927 = queryNorm
            0.3063422 = fieldWeight in 403, product of:
              1.0 = tf(freq=1.0), with freq of:

```

TABELLENVERZEICHNIS

```
1.0 = termFreq=1.0
5.601686 = idf(docFreq=22, maxDocs=2292)
0.0546875 = fieldNorm(doc=403)
0.0011392202 = (MATCH) Math.min of
0.0011392202 = (MATCH) Function for field receivedDate:
0.0011392202 = exp(- MIN[Math.max(Math.abs(1.443660906E12(=doc value)
- 1.455488057445E12(=origin))) - 0.0(=offset), 0)] *
5.73038343716886E-10)
3.4028235E38 = maxBoost
1.0 = queryBoost
)
```

Anhang 5.4: Query Nr. 5 - Explain-Plan des ersten Hits

```

1 Hit: score(4.9056945E-4) | receivedDate(01.10.2015 02-55) | name(Senior
    Java-Entwickler Angebotsimport (m/w)Idealo Internet GmbH - Berlin) |
    id(567264c744aeeb036a01be31)
1 Explain(: 4.9056945E-4 = (MATCH) function score, product of:
    0.431067 = (MATCH) sum of:
        0.19829285 = (MATCH) sum of:
            0.103017226 = (MATCH) weight(content:softwareentwickl in 403)
                [PerFieldSimilarity], result of:
                    0.103017226 = score(doc=403,freq=1.0), product of:
                        0.50549334 = queryWeight, product of:
                            3.7265449 = idf(docFreq=149, maxDocs=2292)
                            0.13564666 = queryNorm
                        0.20379542 = fieldWeight in 403, product of:
                            1.0 = tf(freq=1.0), with freq of:
                                1.0 = termFreq=1.0
                                3.7265449 = idf(docFreq=149, maxDocs=2292)
                                0.0546875 = fieldNorm(doc=403)
                            0.09527563 = (MATCH) weight(content:java in 403) [PerFieldSimilarity],
                                result of:
                                    0.09527563 = score(doc=403,freq=2.0), product of:
                                        0.40878406 = queryWeight, product of:
                                            3.0135949 = idf(docFreq=305, maxDocs=2292)
                                            0.13564666 = queryNorm
                                        0.23307082 = fieldWeight in 403, product of:
                                            1.4142135 = tf(freq=2.0), with freq of:
                                                2.0 = termFreq=2.0
                                                3.0135949 = idf(docFreq=305, maxDocs=2292)
                                                0.0546875 = fieldNorm(doc=403)
                                    0.23277412 = (MATCH) weight(content:testautomatisierung in 403)
                                        [PerFieldSimilarity], result of:
                                            0.23277412 = score(doc=403,freq=1.0), product of:
                                                0.75984997 = queryWeight, product of:
                                                    5.601686 = idf(docFreq=22, maxDocs=2292)
                                                    0.13564666 = queryNorm
                                                0.3063422 = fieldWeight in 403, product of:
                                                    1.0 = tf(freq=1.0), with freq of:
                                                        1.0 = termFreq=1.0
                                                        5.601686 = idf(docFreq=22, maxDocs=2292)
                                                        0.0546875 = fieldNorm(doc=403)
                                            0.0011380353 = (MATCH) Math.min of
                                                0.0011380353 = (MATCH) Function for field receivedDate:
                                                    0.0011380353 = exp(- MIN[Math.max(Math.abs(1.443660906E12(=doc value)
                                                        - 1.455489873391E12(=origin))) - 0.0(=offset), 0]) *
                                                        5.73038343716886E-10)
                                                    3.4028235E38 = maxBoost
                                                1.0 = queryBoost
                                )

```

Anhang 5.5: Query Nr. 6

```
{
  "function_score": {
    "query": {
      "filtered": {
        "query": {
          "bool": {
            "should": [
              {
                "match": {
                  "content": {
                    "query": "Spring Framework",
                    "type": "boolean",
                    "operator": "AND",
                    "boost": 50.0
                  }
                }
              },
              {
                "match": {
                  "content": {
                    "query": "Java",
                    "type": "boolean",
                    "operator": "AND",
                    "boost": 30.0
                  }
                }
              },
              {
                "match": {
                  "content": {
                    "query": "PostgreSQL",
                    "type": "boolean",
                    "operator": "AND",
                    "boost": 20.0
                  }
                }
              },
              {
                "match": {
                  "content": {
                    "query": "Datenbanken",
                    "type": "boolean",
                    "operator": "AND",
                    "boost": 20.0
                  }
                }
              }
            ]
          }
        }
      }
    }
  },
  "filter": {
    "term": {
      "read": false
    }
  }
}
```

```

    }}}
  },
  "functions": [
    {
      "exp": {
        "receivedDate": {
          "scale": "14d"
        }
      }
    }
  ]
}

```

Anhang 5.6: Query Nr. 6 - Explain-Plan von ersten Hit

```

1 Hit: score(5.763883E-4) | receivedDate(01.10.2015 02:55) | name(Senior
    Java-Entwickler Angebotsimport (m/w)Idealo Internet GmbH - Berlin) |
    id(567264c744aeeb036a01be31)
1 Explain(: 5.763883E-4 = (MATCH) function score, product of:
0.39150882 = (MATCH) product of:
0.52201176 = (MATCH) sum of:
0.39096564 = (MATCH) sum of:
0.17041172 = (MATCH) weight(content:spring in 403)
    [PerFieldSimilarity], result of:
0.17041172 = score(doc=403,freq=2.0), product of:
0.5933848 = queryWeight, product of:
    3.7132995 = idf(docFreq=151, maxDocs=2292)
    0.15979987 = queryNorm
0.28718585 = fieldWeight in 403, product of:
    1.4142135 = tf(freq=2.0), with freq of:
        2.0 = termFreq=2.0
    3.7132995 = idf(docFreq=151, maxDocs=2292)
    0.0546875 = fieldNorm(doc=403)
0.22055392 = (MATCH) weight(content:framework in 403)
    [PerFieldSimilarity], result of:
0.22055392 = score(doc=403,freq=3.0), product of:
0.609988 = queryWeight, product of:
    3.8171992 = idf(docFreq=136, maxDocs=2292)
    0.15979987 = queryNorm
0.36157092 = fieldWeight in 403, product of:
    1.7320508 = tf(freq=3.0), with freq of:
        3.0 = termFreq=3.0
    3.8171992 = idf(docFreq=136, maxDocs=2292)
    0.0546875 = fieldNorm(doc=403)
0.06734424 = (MATCH) weight(content:java^30.0 in 403)
    [PerFieldSimilarity], result of:
0.06734424 = score(doc=403,freq=2.0), product of:
0.28894326 = queryWeight, product of:
    30.0 = boost
    3.0135949 = idf(docFreq=305, maxDocs=2292)
    0.0031959976 = queryNorm

```

TABELLENVERZEICHNIS

```
0.23307082 = fieldWeight in 403, product of:
  1.4142135 = tf(freq=2.0), with freq of:
    2.0 = termFreq=2.0
    3.0135949 = idf(docFreq=305, maxDocs=2292)
    0.0546875 = fieldNorm(doc=403)
0.06370189 = (MATCH) weight(content:datenbank^20.0 in 403)
  [PerFieldSimilarity], result of:
0.06370189 = score(doc=403,freq=2.0), product of:
  0.22945254 = queryWeight, product of:
    20.0 = boost
    3.5896857 = idf(docFreq=171, maxDocs=2292)
    0.0031959976 = queryNorm
  0.27762556 = fieldWeight in 403, product of:
    1.4142135 = tf(freq=2.0), with freq of:
      2.0 = termFreq=2.0
      3.5896857 = idf(docFreq=171, maxDocs=2292)
      0.0546875 = fieldNorm(doc=403)
  0.75 = coord(3/4)
0.0014722231 = (MATCH) Math.min of
  0.0014722231 = (MATCH) Function for field receivedDate:
    0.0014722231 = exp(- MIN[Math.max(Math.abs(1.443660906E12(=doc value)
      - 1.455040566297E12(=origin))) - 0.0(=offset), 0]) *
      5.73038343716886E-10)
  3.4028235E38 = maxBoost
  1.0 = queryBoost
)
```

Anhang 5.7: Query Nr. 7

```
{
  "function_score": {
    "query": {
      "filtered": {
        "query": {
          "bool": {
            "should": [
              {
                "function_score": {
                  "query": {
                    "match": {
                      "content": {
                        "query": "Spring Framework",
                        "type": "boolean",
                        "operator": "OR",
                        "boost": 50.0
                      }
                    }
                  }
                },
                "functions": [
                  {
                    "weight": 50.0
                  }
                ]
              }
            ]
          }
        },
        "functions": [
          {
            "weight": 30.0
          }
        ]
      }
    },
    "functions": [
      {
        "weight": 30.0
      }
    ]
  },
  {
    "function_score": {
      "query": {
        "match": {
          "content": {
            "query": "Java",
            "type": "boolean",
            "operator": "OR",
            "boost": 30.0
          }
        }
      },
      "functions": [
        {
          "weight": 30.0
        }
      ]
    },
    {
      "function_score": {
        "query": {
          "match": {
            "content": {
              "query": "PostgreSQL",
```

```

        "type": "boolean",
        "operator": "OR",
        "boost": 20.0
      }}
    },
    "functions": [
      {
        "weight": 20.0
      }
    ]
  },
  {
    "function_score": {
      "query": {
        "match": {
          "content": {
            "query": "Datenbanken",
            "type": "boolean",
            "operator": "OR",
            "boost": 20.0
          }
        }
      },
      "functions": [
        {
          "weight": 20.0
        }
      ]
    }
  }
]
},
"filter": {
  "term": {
    "read": false
  }
}
},
"functions": [
  {
    "exp": {
      "receivedDate": {
        "scale": "14d"
      }
    }
  }
]
}

```

Anhang 5.8: Query Nr. 7 - Explain-Plan von ersten Hit

```

1 Hit: score(0.024797508) | receivedDate(01.10.2015 02-55) | name(Senior
      Java-Entwickler Angebotsimport (m/w)Idealo Internet GmbH - Berlin) |
      id(567264c744aeeb036a01be31)

```

TABELLENVERZEICHNIS

1 Explain(: 0.024797508 = (MATCH) function score, product of:
17.131985 = (MATCH) product of:
22.842648 = (MATCH) sum of:
19.548283 = (MATCH) function score, product of:
0.39096564 = (MATCH) sum of:
0.17041172 = (MATCH) weight(content:spring in 403)
[PerFieldSimilarity], result of:
0.17041172 = score(doc=403,freq=2.0), product of:
0.5933848 = queryWeight, product of:
3.7132995 = idf(docFreq=151, maxDocs=2292)
0.15979987 = queryNorm
0.28718585 = fieldWeight in 403, product of:
1.4142135 = tf(freq=2.0), with freq of:
2.0 = termFreq=2.0
3.7132995 = idf(docFreq=151, maxDocs=2292)
0.0546875 = fieldNorm(doc=403)
0.22055392 = (MATCH) weight(content:framework in 403)
[PerFieldSimilarity], result of:
0.22055392 = score(doc=403,freq=3.0), product of:
0.609988 = queryWeight, product of:
3.8171992 = idf(docFreq=136, maxDocs=2292)
0.15979987 = queryNorm
0.36157092 = fieldWeight in 403, product of:
1.7320508 = tf(freq=3.0), with freq of:
3.0 = termFreq=3.0
3.8171992 = idf(docFreq=136, maxDocs=2292)
0.0546875 = fieldNorm(doc=403)
50.0 = (MATCH) Math.min of
50.0 = (MATCH) product of:
1.0 = constant score 1.0 - no function provided
50.0 = weight
3.4028235E38 = maxBoost
1.0 = queryBoost
2.0203273 = (MATCH) function score, product of:
0.06734424 = (MATCH) weight(content:java^30.0 in 403)
[PerFieldSimilarity], result of:
0.06734424 = score(doc=403,freq=2.0), product of:
0.28894326 = queryWeight, product of:
30.0 = boost
3.0135949 = idf(docFreq=305, maxDocs=2292)
0.0031959976 = queryNorm
0.23307082 = fieldWeight in 403, product of:
1.4142135 = tf(freq=2.0), with freq of:
2.0 = termFreq=2.0
3.0135949 = idf(docFreq=305, maxDocs=2292)
0.0546875 = fieldNorm(doc=403)
30.0 = (MATCH) Math.min of
30.0 = (MATCH) product of:
1.0 = constant score 1.0 - no function provided

TABELLENVERZEICHNIS

```
    30.0 = weight
    3.4028235E38 = maxBoost
    1.0 = queryBoost
1.2740378 = (MATCH) function score, product of:
    0.06370189 = (MATCH) weight(content:datenbank^20.0 in 403)
        [PerFieldSimilarity], result of:
    0.06370189 = score(doc=403,freq=2.0), product of:
    0.22945254 = queryWeight, product of:
        20.0 = boost
        3.5896857 = idf(docFreq=171, maxDocs=2292)
        0.0031959976 = queryNorm
    0.27762556 = fieldWeight in 403, product of:
        1.4142135 = tf(freq=2.0), with freq of:
            2.0 = termFreq=2.0
            3.5896857 = idf(docFreq=171, maxDocs=2292)
            0.0546875 = fieldNorm(doc=403)
20.0 = (MATCH) Math.min of
    20.0 = (MATCH) product of:
        1.0 = constant score 1.0 - no function provided
        20.0 = weight
        3.4028235E38 = maxBoost
        1.0 = queryBoost
0.75 = coord(3/4)
0.0014474393 = (MATCH) Math.min of
    0.0014474393 = (MATCH) Function for field receivedDate:
        0.0014474393 = exp(- MIN[Math.max(Math.abs(1.443660906E12(=doc value)
            - 1.455070193609E12(=origin))) - 0.0(=offset), 0)] *
            5.73038343716886E-10)
        3.4028235E38 = maxBoost
        1.0 = queryBoost
)
```

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Datum: 15. Februar 2016, Berlin

Unterschrift: